Variants of Group Signatures and Their Applications

Lydia Garms

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Information Security Group
School of Engineering, Physical and Mathematical Sciences
Royal Holloway, University of London

2020

Declaration

These doctoral studies were conducted under the supervision of Professor Keith Martin.

The work presented in this thesis is the result of original research I conducted, in collaboration with others, whilst enrolled in the School of Mathematics and Information Security as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Lydia Garms

27th February 2020

Abstract

In public–key cryptography each entity generates a public and secret key. The public key is published alongside the user's identity, whereas the secret key is kept private. Signature schemes allow a signer with their secret key to "sign" a message producing a digital signature. This signature can be publicly verified and ensures that the message originated from the signer associated with the public key. Group signatures allow users to sign on behalf of a group of users. Essentially, this ensures that the message originates from any member of the group. However their identity within this group is not revealed, except to a trusted opener who holds the opening secret key. In this thesis we introduce several variants of group signature schemes for two applications.

The first application is reputation systems, which assign a user or item a reputation value that can be used to evaluate trustworthiness. Examples of this include Amazon, which rates sellers out of 5, and AirBnb which rates properties and users out of 5. In a reputation system a user often has multiple items associated with them; for example, on Amazon a user may sell several products. We introduce a new cryptographic model, whereby reputation values are given to users, as opposed to existing cryptographic models where reputation values are given to items. To allow for user privacy, a user's items cannot be linked. We prove this model can be achieved using two variations of a group signatures scheme, with low additional efficiency cost given the extra functionality.

The collection of user data, such as health care records or other personal data, for processing is our second application. Group signatures allow user data to be collected while preserving the user's privacy but still ensuring it originates from a group member. However, the correlation of data by user is useful for processing data. Therefore, the linkability, i.e. whether signatures can be linked by user, must balance utility and privacy. We introduce a new variant of group signature scheme that provides a more flexible and privacy-friendly form of linkability. When created, all signatures are fully unlinkable, but can be made linkable via a centrally trusted entity known as the *converter*. We formally define the requirements for this new type of group signature scheme and provide an efficient instantiation that provably satisfies these requirements.

The previous model captures a setting where the entity collecting and processing data is the same. Therefore the data collector can be assumed to only submit honest input to the converter. The outputs of the converter do not provide any assurance that data originated from a group member. We extend the previous model to remove this assumption and allow authentication to be preserved after data is converted. In order to provide a provably secure construction we introduce: *commuting group signatures*. These signatures lift the idea from the existing work of commuting signatures to the setting of group signatures. We formally introduce these signatures and show how they can be realised.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly I would like to thank my supervisor Keith for guiding me through my PhD, as well as for all of the helpful feedback and advice I have received. I have also learned a great deal from working with my other co authors. I would like to thank Siaw-Lynn and Liz for all the time and support they have given me, and Anja for giving me the opportunity to work with her during my internship, as well as the continued guidance during the rest of my PhD studies. I would like to thank the EPSRC grant (EP/K035584/1) for funding my PhD and IBM Research for funding my internship.

Thanks to everyone in the department for being such good company. I have really enjoyed being part of a centre for doctoral training. My CDT cohort, particularly Ben, Nick and Ivan as well as others from different cohorts, have been a great source of support throughout the PhD. Thanks to all previous occupants of office 253 for the many tea breaks, boilerhouse trips, SCR lunches and pub quizzes. In particular, thanks to Keele for feeding me vegetables and generally being amazing, to Reynold for giving my day a (strictly enforced) social structure, and to Nick (2) for his youthful approach to life. Being part of the Wisdom group really improved my experience in the department and I am grateful to Thyla, Sheila, Thalia and Rachel for their work in establishing this.

I am very grateful for the support I have received throughout my PhD studies from my family and friends. In particular, thanks to Harriet and Lucy for always being there to cheer me up at weekends, as well as my Mum and Dad for their unending support. Finally thanks to Amit for being someone to bounce ideas off of, for consoling me when things weren't going to plan, for waking me up in the mornings with a cup of tea and much much more.

Chapter 1

Introduction

Contents

This chapter provides the motivation for the work in this thesis and an overview of the structure.

1.1 Motivation

Group signature schemes [49] allow group members to sign on behalf of a group, without revealing their identity within this group. A *group signature* ensures that a message originates from a member of the group. To ensure accountability, a trusted *opener* has the power to de-anonymise signatures with the opening secret key. In this thesis we will introduce variants of group signature schemes for two different applications.

Our first application is *centralised reputation systems* [7, 18, 58, 70, 106, 112, 128], where a user or item is allocated a *reputation value* representing trustworthiness or merit. Centralised schemes are managed by a trusted entity who forms reputation values. However forming reputation values is often at odds with privacy. If reputation values are formed from *feedback* given by other users, then the privacy of this feedback should be ensured, whilst also ensuring that feedback is not given in an unfair way. For example a user could unfairly influence another's reputation by giving a large amount of positive or neg-

ative feedback on the same subject. Also to form a reputation value on a user's entire behaviour, their activities must be linked together. This linkability could potentially be used to de-anonymise users.

In order to address this, cryptographic models have been proposed to capture the key security and privacy properties that a reputation system should satisfy [7, 18, 58]. Different cryptographic techniques have been used to provide instantiations that achieve these models. Group signature schemes have been used to allow feedback to be provided while avoiding multiple feedback on the same item [18, 58]. They have also been used to enable the formation of reputation values whilst ensuring the privacy of users' behaviour [70, 106].

Contribution. In this thesis we establish a cryptographic model for a centralised reputation system in a setting where user behaviour is *unlinkable* and reputation is given to users instead of items. This ensures that users are accountable for all their behaviour within the system. For example, in the context of a car pooling app, we ensure that a user's journeys (or items) cannot be linked together. However, we still allow for a user's reputation to be based on all of their activities on the app, so that it accurately reflects their overall behaviour. We show that this model can be achieved by using two variants of group signature schemes. The first variant allows reputation to be bound to group signatures, so that reputation can be proved alongside group membership. The second variant is based on an existing primitive, *direct anonymous attestation* [29].

Our second application is the collection and processing of data for privacy enhancing applications. Group signatures enable data to be stored authenticated, whilst also preserving a user's privacy. However there should be a balance between privacy and utility: the correlation of data by user is often necessary for data processing. For example, several high value measurements of blood pressure over many users may have very different implications than if they were all associated with a single user. Often the exact reason for data processing is not clear when data is collected. *Data collectors* gather large amounts of data but will only use small subsets when necessary for particular applications. For example, public Wi-Fi data was accidentally captured by Google Street View cars but later used to improve Google's location services.

The *opener*, a trusted entity with the opening secret key, could de-anonymise signatures to

allow for processing. However, this is potentially privacy intrusive, as the opener has the power to de-anonymise all signatures. Alternative variants of group signatures have been proposed which, instead of the opening functionality, allow signatures to be linked. This means that signatures do not have to be fully de-anonymised, yet the correlation of data by user can still be determined. *Controlled linkability* [79, 80, 120] allows for the linking of signatures by a trusted entity with a *linking secret key*. However, this is still privacy invasive since the linking entity can obtain information about the linking of signatures. Also it does not scale well, as each pair of signatures must be linked. This means linking a set of n signatures would require $\binom{n}{2}$ operations. *User controlled linkability* [29] allows a user to choose a *basename*, as well as a message to sign. Signatures on the same basename can be publicly linked. However the linking can be determined by all, not just a trusted entity. Also the signatures that need to be linked must be determined at the point of data collection, which is not flexible enough. This is because, the purpose of the data may not be known until later.

Contribution. In this thesis we firstly introduce a new variant of group signatures: *group signatures with selective linkability*. Data is stored authenticated and unlinkable by the *data collector*. We refer to the part of a signature that provides linkability as a *pseudonym*. When a subset of pseudonyms, are processed, they can be linked in a controlled way by a trusted central entity known as the *converter*. Specifically, this linking should be performed *blindly*, so that the messages/ pseudonyms that are being linked, and their linkage is not revealed to the converter. This linking is *non-transitive*, meaning pseudonyms can only be linked within the same query to the converter. This ensures that a list of linked pseudonyms cannot be gradually built up by the data collector. We assume both the data collector, and converter are *honest–but–curious*, i.e. we assume they follow the protocol honestly, but will attempt to learn all possible information. Therefore, only correctly formed messages and pseudonyms are input to the converter.

Secondly, we introduce an extended model, for a setting where the data collector is split into a *data lake* and *data processor*. A data lake is an entity that has collected a large amount of user data, which they could then sell or pass on to data processors if a use for this data is found. For example, groups of hospitals could collect anonymised healthcare records of patients stored in a data lake. Researchers could then request relevant data for analysis. The data processor can no longer trust that correctly formed pseudonyms

were input to the converter. Therefore signatures as well as pseudonyms must be input to the converter. To ensure that authentication is preserved, the converter must also output signatures. Finally, in order to achieve this model, we introduce *commuting group signatures* as a building block. These allow both messages and signatures to be blinded, whilst still allowing for public verifiability, similarly to commuting signatures [66]. Blinded signatures can still be opened to reveal a blinded identity. Therefore, blinded group signatures can now be input to the converter who can verify and blindly open them.

1.2 Chapter Overviews

Chapter 2 presents preliminaries used throughout this thesis. This includes an introduction to the notation used and core definitions.

In Chapter 3 we focus on the first application of reputation systems. We provide a cryptographic security model for a centralised reputation system where reputation values are given to users instead of items. We then present two building blocks and show how they can be used to build a construction for such a reputation system. These are a direct anonymous attestation scheme [29], and a modification to an existing group signature scheme that allows reputation to be proved alongside group membership. We then prove the security of this construction in our model, and evaluate how susceptible it is to conventional attacks against reputation systems outside the scope of our cryptographic model. We provide a concrete instantiation of our construction and analyse the efficiency of this in comparison to other relevant reputation systems.

In Chapter 4 we consider our second application of data collection/ processing. We provide a security model for our group signatures with selective linkability, defining the security and privacy properties required. We then present a construction and prove the security of this in the model. This construction makes use of *ElGamal encryption* [59], *BBS+ signatures* [9] and *signature proofs of knowledge* [46]. We then provide a concrete instantiation for these building blocks and evaluate the efficiency of this in terms of the sizes of pseudonyms/ signatures, and the number of operations required.

In Chapter 5 we introduce commuting group signatures, which will be used as a building block to achieve the extended model for group signatures with selective linkability defined

in Chapter 6. We first present a formal security model for this new primitive, including syntax and the necessary security and privacy properties. The standard security requirements for group signatures should apply, but now must hold for both standard and blinded signatures. We also must ensure the security of the blinding of messages and signatures. We then provide a construction, making use of *malleable proof systems* [45], *automorphic signatures* [65], and ElGamal encryption. We prove our construction is secure, and then provide a concrete instantiation of these building blocks. We compute the sizes of these signatures in terms of the number of group elements.

In Chapter 6 we provide an extended security model for group signatures with selective linkability, which takes into account the split data lake/ data processor. We provide updated syntax to take into account that signatures are now input to the blind and convert algorithms. We extend our security requirements to so that we no longer assume that all inputs to convert from the data lake are correctly formed. We also ensure that the unforgeability properties now hold for blinded and converted signatures. We then present a construction making use of a commuting group signature scheme, as given in Chapter 5, as a building block. The construction also makes use of a signature proof of knowledge and a standard signature scheme. We then prove this construction secure under the extended security model. We show that the concrete instantiation of commuting group signatures given in Chapter 5 is compatible with our construction, and also provide an instantiation of the other building blocks used. We analyse the efficiency of this in comparison to the instantiation given in Chapter 4.

1.3 Publications

Chapter 3 is based on the paper "A New Approach to Modelling Centralised Reputation Systems", which was joint work with Elizabeth A. Quaglia. This was presented at AFRICACRYPT 2019 [71]. Preliminary work on this topic was included in the paper "Reputation Schemes for Pervasive Social Networks with Anonymity", which was joint work with Keith Martin and Siaw–Lynn Ng. This was presented at PST 2017 [70].

Chapter 4 is based on the paper "Group Signatures with Selective Linkability", which was joint work with Anja Lehmann. This was presented at PKC 2019 [69].

Chapters 5 and 6 are based on the paper "Convertible Group Signatures – Stronger Security and Preserved Verifiability", which was joint work with Anja Lehmann. This is currently under submission.

Chapter 2

Preliminaries

Contents

This chapter introduces the standard notation, key building blocks, and definitions that will be used throughout this thesis. For a comprehensive background on these notions, see [82].

2.1 Notation

We first provide some basic notation and group theory that will be used throughout the thesis.

We denote the set of integers as $\mathbb{Z}$, the set of natural number as $\mathbb{N}$ and the set of real numbers as $\mathbb{R}$. A positive integer is a *prime* if it only has two divisors: 1 and itself. Let $\mathbb{Z}_p^*$

denote the set $\{1, .., p-1\}$. We denote the set of consecutive integers $\{i, ..., j\}$ for integers $i \leq j$, as $[i, j]$, and $[n]$ denotes the set $\{1, ..., n\}$ for $n > 1$. We denote the empty set as $\emptyset$.

A function, $f : \mathbb{N} \to \mathbb{R}$, is *negligible* on its input, if for every positive polynomial p, there exists an N such that for all integers $n > N$, $f(n) < \frac{1}{p(n)}$. Throughout this thesis, we shall denote an arbitrary negligible function by $\mathsf{negl}()$.

We denote assigning the value x to the variable y as $y \leftarrow x$, and $y \leftarrow\!\!\$\, S$ denotes choosing an element from a finite set S uniformly at random and assigning it to the variable y. If A is a deterministic function, $y \leftarrow A(x_1, ...x_n)$ denotes assigning the result of running A on the inputs x_1 to x_n to an output y. If A is instead a probabilistic algorithm, $y \leftarrow\!\!\$\, A()$ denotes assigning the output of A to the variable y. We let $\perp$ denote a failure symbol, and PPT denote *probabilistic polynomial–time*.

2.2 Provable Security of Cryptography

In cryptography it is vital to precisely define security for cryptographic primitives, because otherwise it is not possible to precisely evaluate the security of the construction.

One way of defining security is using *security games*, which will be the method we use throughout this thesis. The games are designed to model real life security, incorporating the threat model by the input given to the adversary, and the security goals by the conditions the adversary must satisfy to win. In a security game a challenger interacts with a probabilistic polynomial–time adversary $\mathcal{A}$. The challenger provides the adversaries inputs and receives outputs in return. In some cases the adversary may keep track of a *state* throughout, so can "remember" past inputs and computations. At the end of the game the challenger decides if the adversary has won or lost based on what they have output. We say that if any adversary can only "win" with negligible probability then the cryptographic primitive is secure. The adversaries are often given access to *oracles* for particular functions. These allow the adversary to submit queries to the challenger who performs some function on this input, alongside some other private input, and returns the output of this function to the adversary.

Security can also be defined using *simulation* based security definitions. In this case, an

ideal functionality is given for the cryptographic primitive and we say a construction is secure if it is indistinguishable to an adversary from the ideal functionality.

In order to ensure that a construction satisfies the security requirements for a cryptographic primitive, *proofs of security* must be provided that this is the case given a set of assumptions. These proofs of security often take the form of reductions from the security of the scheme to a cryptographic hardness assumption.

2.3 The Discrete Logarithm problem

We let $\mathcal{G}$ denote a group generation algorithm that on input a security parameter 1^τ outputs a description of a cyclic group along with its prime order p and a generator $g \in \mathbb{G}$. The size of the group output will depend on the security parameter τ input. We require that the group operation can be computed efficiently. For each $h \in \mathbb{G}$ there is a unique x in $\mathbb{Z}_p$, such that $h = g^x$. We call this x the *discrete logarithm* of h with respect to g. The *discrete logarithm problem* in a cyclic group $\mathbb{G}$ is, for a uniformally chosen h, to compute the discrete logarithm of h with respect to g. We say the discrete logarithm problem is hard with respect to $\mathcal{G}$, if for all PPT algorithms $\mathcal{A}$, the probability that they solve the discrete logarithm problem for the group output by $\mathcal{G}$ with input 1^τ, is negligible in τ. The *discrete logarithm assumption* is the assumption that there exists $\mathcal{G}$ for which the discrete logarithm problem is hard.

2.3.1 The Diffie–Hellman problems

The *Computational Diffie–Hellman assumption*, which we will refer to as the CDH assumption, and the *Decisional Diffie–Hellman assumption*, which we will refer to as the DDH assumption, are two assumptions that are related (but not equivalent) to the Discrete Logarithm assumption.

The *Computational Diffie–Hellman problem* is given a prime p order group $\mathbb{G}$, a generator g and uniformally chosen group elements $h_1, h_2 \in \mathbb{G}$, compute $g^{x_1 x_2}$ where x_1, x_2 are the discrete logarithms of h_1, h_2 with respect to g .

The *Decisional Diffie–Hellman problem* is given a prime p order group $\mathbb{G}$, a generator g and uniformally chosen group elements $h_1, h_2 \in \mathbb{G}$, to distinguish between $g^{x_1 x_2}$ where x_1, x_2 are the discrete logarithms of h_1, h_2 with respect to g and a uniform group element.

The *random self–reduction for DDH* [105, 122] randomises a DDH problem instance so that if the input is a random DDH 3–tuple then the output is a random DDH 3–tuple, and otherwise the output is a random 3–tuple. In [127] an *expanded self–reduction* is given that will be used in this thesis. If the input is a random DDH 3–tuple (x, y, z) then the output is two random DDH 3–tuples (x', y', z') and (x'', y', z'') with y' in common, and otherwise (x', x'', y', z', z'') is a random 5–tuple. This can be generalised so that the output is either k random DDH tuples with y' in common, or a random $2k + 1$–tuple.

2.3.2 Bilinear Maps

Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be cyclic groups of prime order p. A *bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ must satisfy the following conditions: *bilinearity*, i.e., $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$; *non-degeneracy*, i.e., for all generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, $e(g_1, g_2)$ generates $\mathbb{G}_T$; and *efficiency*, i.e., there exists an efficient algorithm $\mathcal{G}(1^\tau)$ that outputs a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, and an efficient algorithm to compute $e(a, b)$ for all $a \in \mathbb{G}_1$, $b \in \mathbb{G}_2$. Bilinear maps are also referred to as *pairings*.

In [67] pairings are classified into three different types. For *type–1* pairings, $\mathbb{G}_1 = \mathbb{G}_2$. For *type–2* pairings, $\mathbb{G}_1 \neq \mathbb{G}_2$, but there is an efficiently computable homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$. For *type–3* pairings, $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no efficiently computable homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$.

We use type–3 pairings [67] in this work; i.e., we do not assume $\mathbb{G}_1 = \mathbb{G}_2$ or the existence of an homomorphism between both groups in our scheme and security proofs. The advantage of type–3 pairings is that they enjoy the most efficient curves, when balancing the cost of pairings and group operations, the size of the representation of an element of $\mathbb{G}_2$ and the flexibility of parameter choice [47, 67].

2.3.3 q-Strong Diffie–Hellman Assumption

The *q-Strong Diffie–Hellman assumption* which we will refer to as the q-SDH assumption is another hardness assumption related to the discrete logarithm assumption.

There are two versions of the q-Strong Diffie–Hellman assumption. The first version, given by Boneh and Boyen in [19], is defined in a type–1 or type–2 pairing setting. We will refer to this version as the Eurocrypt version. It is stated as follows:

Definition 2.1 (q–SDH assumption (Eurocrypt Version))**.** In $(\mathbb{G}_1, \mathbb{G}_2)$, with an homomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$: given $(S_1, S_2, S_2^{\chi}, S_2^{(\chi)^2}, ..., S_2^{(\chi)^q})$ such that $S_1 \in \mathbb{G}_1, S_2 \in \mathbb{G}_2, S_1 = \psi(S_2)$, output $(S_1^{\frac{1}{\chi+x}}, x) \in \mathbb{G}_1 \times \mathbb{Z}_p \backslash \{-\chi\}$.

We use the second version of that definition that supports type–3 pairings and was stated in the journal version of their paper [20]. We will refer to this version as the JoC version. It is stated as follows:

Definition 2.2 (q-SDH assumption (JoC version))**.** Given $(g_1, g_1^{\chi}, g_1^{(\chi)^2}, ..., g_1^{(\chi)^q}, g_2, g_2^{\chi})$ such that $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, output $(g_1^{\frac{1}{\chi+x}}, x) \in \mathbb{G}_1 \times \mathbb{Z}_p \backslash \{-\chi\}$.

2.3.4 LRSW Assumption

Another hardness assumption related to the discrete logarithm problem we will use in this thesis is the *LRSW assumption* [97].

Definition 2.3 (LRSW assumption)**.** Let an adversary be given $(\mathbb{G}_1, \mathbb{G}_2)$, with generators G_1, G_2 of $\mathbb{G}_1$, $\mathbb{G}_2$ respectively, $X = G_2^x$, $Y = G_2^y$ for some $x, y \in \mathbb{Z}_p^*$ and access to an oracle that when f is input, outputs $(A, A^{\beta}, A^{\alpha} A^{f\alpha\beta})$ where $A = G_2^r$ for uniform $r \in \mathbb{Z}_p^*$. It is hard for the adversary to output (f, A, B, C) such that $A \in \mathbb{G}_1$, $B = A^{\beta}, C = A^{\alpha} A^{f\alpha\beta}$, and f has not been queried to the oracle.

2.4 Random Oracle Model

In this thesis we will make use of the *random oracle model* [13] as an assumption. The random oracle model is a powerful tool that allows many practical schemes to be proven

secure.

The random oracle model firstly assumes that a hash function can be modelled in security proofs by a black box that on receiving an input x outputs y. If x has been input previously, then the same y should be output. If x has not been input previously, then y should be chosen uniformally at random, and (x, y) should be stored for future queries. If an adversary queries x to $\mathcal{H}$ then the reduction in the proof of security can see the query and learn x, which is referred to as *extractability*. The reduction can also set the value of $\mathcal{H}(x)$ to a value of its choice, provided it is correctly distributed, which is referred to as *programmability*.

The value of proofs based on the random oracle model is still under debate within the cryptographic community. Examples of cryptographic schemes that can be proved secure under the random oracle, but where any practical instantiation can be shown to be insecure were found in [42]. However these schemes were contrived, and so far there have been no successful "real world" attacks on schemes proved secure under the random oracle model. When instantiated properly with a suitable hash function, proofs under the random oracle are still seen as strong evidence that a cryptographic construction will resist attacks in practise [31].

2.5 Basic Cryptographic Primitives

2.5.1 Encryption Schemes

A *public–key encryption scheme* consists of the following probabilistic polynomial–time algorithms:

- $\mathsf{EncKGen}(1^\tau)$: takes as input the security parameter 1^τ and outputs the public key pk and secret key sk.
- $\mathsf{Enc}(pk, m)$: takes as input the public key pk and a *message* m from the message space, and outputs a *ciphertext* c.
- $\mathsf{Dec}(sk, c)$: takes as input the secret key sk and a ciphertext c, and outputs a message

m or a decryption failure $\perp$.

Correctness. We say the encryption scheme is *correct* if for all messages m in the message space, the probability that $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) \neq m$ is negligible in τ. We say an encryption scheme is *perfectly correct* if, for all messages m in the message space, the probability that $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) \neq m$ is 0.

2.5.1.1 Security Against Chosen–Plaintext Attacks

Security against chosen–plaintext attacks captures that an adversary should not be able to distinguish between encryptions of two messages chosen by the adversary.

We now present the security game corresponding to the *indistinguishable encryptions under a chosen–plaintext attack* (ind-cpa) requirement.

Experiment: $\mathbf{Exp}^{ind-cpa-b}_{\mathcal{A},\Pi}(\tau)$

$(pk, sk) \leftarrow\!\!\$\ \mathsf{EncKGen}(1^\tau), (m_0, m_1, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{choose}, pk)$

$c \leftarrow\!\!\$\ \mathsf{Enc}(pk, m_b), b^* \leftarrow\!\!\$\ \mathcal{A}(\mathsf{st}, \mathsf{guess}, c)$

return b^*

Definition 2.4. A public–key encryption scheme Π has indistinguishable encryptions under a chosen–plaintext attack if, for all probabilistic polynomial–time adversaries $\mathcal{A}$, the following advantage is negligible in τ: $|\Pr[\mathbf{Exp}^{ind-cpa-0}_{\mathcal{A},\Pi}(\tau) = 1] - \Pr[\mathbf{Exp}^{ind-cpa-1}_{\mathcal{A},\Pi}(\tau) = 1]|$.

2.5.1.2 Security Against Chosen–Ciphertext Attacks

Security against chosen–ciphertext attacks captures that an adversary should not be able to distinguish between encryptions of two messages, even when the adversary is able to obtain decryptions of arbitrary ciphertexts. This ensures that the encryption scheme is non–malleable [11]. This means that, given an encryption c of a message m, an adversary cannot transform this into a new ciphertext, which is an encryption of a known function of the original message.

We now present the security game corresponding to the *indistinguishable encryptions under a chosen–ciphertext attack* requirement.

$\mathsf{DEC}(c')$

if $c' = c$ **return** $\perp$ **else return** $m' \leftarrow \mathsf{Dec}(sk, c')$

Experiment: $\mathbf{Exp}^{ind-cca2-b}_{\mathcal{A},\Pi}(\tau)$

$(pk, sk) \leftarrow\!\!\$\ \mathsf{EncKGen}(1^\tau), (m_0, m_1, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{DEC}}(\mathsf{choose}, pk)$

$c \leftarrow\!\!\$\ \mathsf{Enc}(pk, m_b), b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{DEC}}(\mathsf{st}, \mathsf{guess}, c)$

return b^*

Definition 2.5. A public–key encryption scheme Π has indistinguishable encryptions under a chosen–ciphertext attack if, for all probabilistic polynomial–time adversaries $\mathcal{A}$, the following advantage is negligible in τ: $|\Pr[\mathbf{Exp}^{ind-cca2-0}_{\mathcal{A},\Pi}(\tau) = 1] - \Pr[\mathbf{Exp}^{ind-cca2-1}_{\mathcal{A},\Pi}(\tau) = 1]|$.

2.5.2 ElGamal Encryption

We will use the *ElGamal* encryption scheme [59] in this thesis. This encryption scheme has public parameters $(\mathbb{G}, g, p)$. We now present algorithms ElgGen, ElgEnc, ElgDec.

$\mathsf{ElgGen}(1^\tau)$

$sk \leftarrow\!\!\$\ \mathbb{Z}^*_p, pk \leftarrow g^{sk}$

return (pk, sk)

$\mathsf{ElgEnc}(pk, m)$

$r \leftarrow\!\!\$\ \mathbb{Z}^*_p$ **return** $c \leftarrow (g^r, pk^r m)$

$\mathsf{ElgDec}(sk, (c_1, c_2))$

return $m \leftarrow c_2 c_1^{-sk}$

Assuming the DDH assumption holds in $\mathbb{G}$, ElGamal encryption is chosen–plaintext secure [59].

Later in this thesis, we will use the *homomorphic* property of ElGamal; i.e., if $C_1 \in \mathsf{ElgEnc}(pk, m_1)$ and $C_2 \in \mathsf{ElgEnc}(pk, m_2)$, then $C_1 \odot C_2 \in \mathsf{ElgEnc}(pk, m_1 \cdot m_2)$.

We further use that ElGamal ciphertexts $c = \mathsf{ElgEnc}(pk, m)$ are *publicly re-randomisable*; i.e., the re-randomised version c' of c looks indistinguishable from a fresh encryption of

the underlying plaintext m. The following procedure clearly satisfies this:

Re-randomisation: On input $(pk, (c_1, c_2))$, $r' \leftarrow_\$ \mathbb{Z}_p^*$ and output $(c_1 g^{r'}, c_2 pk^{r'})$.

2.5.3 Signature Schemes

A *signature scheme* consists of the following probabilistic polynomial–time algorithms.

- $\mathsf{SigKGen}(1^\tau)$: takes as input the security parameter 1^τ and outputs the public verification key vk and secret signing key sk.
- $\mathsf{Sign}(sk, m)$: takes as input the signing key sk and a message m, and outputs a signature σ.
- $\mathsf{Verify}(vk, m, \sigma)$: takes as input the verification key vk, a message m, and signature σ, and outputs 1 if σ is valid with respect to vk, m, and 0 otherwise.

Correctness. We say a signature scheme is *correct* if for all messages m in the message space, the probability that $\mathsf{Verify}(vk, m, \mathsf{Sign}(sk, m)) \neq 1$ is negligible in τ. We say a signature scheme is perfectly correct if for all messages m in the message space, the probability that $\mathsf{Verify}(vk, m, \mathsf{Sign}(sk, m)) \neq 1$ is 0.

2.5.3.1 Defining the Security of Signature Schemes

The security requirements capture that an adversary cannot produce a signature on a message of their choice without knowledge of the secret key, even when it can see signatures of other messages.

We now present the security game corresponding to the *existential unforgeability under an adaptive chosen–message attack* (EUF-cma) requirement. The adversary is given the verification key vk and a signing oracle for secret key sk. The signing oracle is input a message m', and keeps track of inputs in the list M. The adversary must output a valid signature for a message m not stored in M.

$\mathsf{SIGN}(m')$

$M \leftarrow M \cup \{m'\}$ **return** $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, m')$

Experiment: $\mathbf{Exp}^{sig-forge}_{\mathcal{A},\Pi}(\tau)$

$(vk, sk) \leftarrow\!\!\$\ \mathsf{SigKGen}(1^\tau), M \leftarrow \emptyset, (m, \sigma) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SIGN}}(vk)$
if $m \in M$ or $\mathsf{Verify}(vk, m, \sigma) \neq 1$ **return** 0 **else return** 1

Definition 2.6. A signature scheme Π is existentially unforgeable under an adaptive chosen–message attack if, for all probabilistic polynomial–time adversaries $\mathcal{A}$, there is a negligible function negl such that $\Pr[\mathbf{Exp}^{sig-forge}_{\mathcal{A},\Pi}(\tau) = 1] \leq \mathsf{negl}(\tau)$.

There is also a strengthened version of this security notion: *strong existential unforgeability under an adaptive chosen–message attack*. We now present the security game for this requirement.

$\mathsf{SIGN}(m')$

$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, m'), M \leftarrow M \cup \{m', \sigma\}$ **return** σ

Experiment: $\mathbf{Exp}^{sig-strongforge}_{\mathcal{A},\Pi}(\tau)$

$(vk, sk) \leftarrow\!\!\$\ \mathsf{SigKGen}(1^\tau), M \leftarrow \emptyset, (m, \sigma) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SIGN}}(vk)$
if $(m, \sigma) \in M$ or $\mathsf{Verify}(vk, m, \sigma) \neq 1$ **return** 0 **else return** 1

Definition 2.7. A signature scheme Π is strongly existentially unforgeable under an adaptive chosen–message attack if, for all probabilistic polynomial–time adversaries $\mathcal{A}$, there is a negligible function negl such that $\Pr[\mathbf{Exp}^{sig-strongforge}_{\mathcal{A},\Pi}(\tau) = 1] \leq \mathsf{negl}(\tau)$.

2.5.4 Structure Preserving Signatures Schemes

A *structure preserving signature scheme* [5] over a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ is an EUF–cma secure signature scheme such that the verification keys, messages and signatures are elements of $\mathbb{G}_1$ and $\mathbb{G}_2$, and the verification predicate is a conjunction of pairing-product equations over the verification key, the message and the signature.

2.5.5 Camenisch-Lysyanskaya Signatures

We will use the *Camenisch–Lysyanskaya* (CL) signature scheme [39] in Chapter 3. The public parameters of the signature scheme are the bilinear group: $\mathsf{param}_{\mathsf{CL}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$. We now present the algorithms CLKGen, CLSign, CLVerify.

$\mathsf{CLKGen}(\mathsf{param}_{\mathsf{CL}})$

$x \leftarrow_\$ \mathbb{Z}_p^*, y \leftarrow_\$ \mathbb{Z}_p^*, sk \leftarrow (x, y), X \leftarrow G_2^x, Y \leftarrow G_2^y, vk \leftarrow (X, Y)$ **return** (vk, sk)

$\mathsf{CLSign}((x, y), m)$

$A \leftarrow_\$ \mathbb{G}_1,$ **return** $\sigma \leftarrow (A, A^y, A^x A^{mxy})$

$\mathsf{CLVerify}((X, Y), m, (A, B, C))$

if $e(A, Y) = e(B, G_2)$ and $e(AB^m, X) = e(C, G_2)$ **return** 1 **else return** 0

The CL signature scheme is existentially unforgeable under a chosen–message attack under the LRSW assumption [39].

2.5.6 BBS+ Signatures

We will use the *BBS+* signature scheme in Chapter 4. This signature scheme was given by Au et al. [9], and inspired by the *BBS* group signature scheme introduced in [21]. The public parameters of the signature scheme are the bilinear group: $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2)$. We now present the algorithms BBSGen+, BBSSign+, BBSVerify+.

$\mathsf{BBSGen}+(1^\tau)$

$h_1 \leftarrow_\$ \mathbb{G}_1, h_2 \leftarrow_\$ \mathbb{G}_1, x \leftarrow_\$ \mathbb{Z}_p^*, w \leftarrow G_2^x, sk \leftarrow x, pk \leftarrow (w, h_1, h_2)$ **return** (pk, sk)

$\mathsf{BBSSign}+(x, m)$

$s_1 \leftarrow_\$ \mathbb{Z}_p, s_2 \leftarrow_\$ \mathbb{Z}_p, A \leftarrow (G_1 h_1^{s_2} h_2^m)^{\frac{1}{s_1 + x}}$ **return** (A, s_1, s_2)

$\mathsf{BBSVerify}+((w, h_1, h_2), m, (A, s_1, s_2))$

if $A \in \mathbb{G}_1$ and $e(A, wG_2^{s_1}) = e(G_1 h_1^{s_2} h_2^m, G_2)$ **return** 1 **else return** 0

Originally, Au et al. [9] proved the BBS+ signature scheme existentially unforgeable under chosen–message attacks under the Eurocrypt version of the q-SDH assumption, making use of the homomorphism between the groups in the security proof. However, in our

security proofs, we will make use of techniques from [32] which prove the unforgeability of BBS+ signatures in the type–3 setting.

2.6 Zero–Knowledge Proofs of Knowledge

Intuitively, *zero–knowledge* proofs allow a prover to interact with a verifier demonstrating that a proposition holds, without revealing any extra information. In fact, if the proposition is true, the verifier might as well have simulated the interaction on their own. A *proof of knowledge* proves not only that a proposition holds but also that the prover knows a valid witness of this.

We define a zero–knowledge proof of knowledge (ZKPoK) for a *language* L, which consists of a set of *instances* for which there exist a valid *witness*, defined by the relation $\mathcal{R}$. More formally $L = \{x : \exists w \text{ s.t } (x, w) \in \mathcal{R}\}$.

Definition 2.8 (Zero–Knowledge Proof of Knowledge)**.** An interactive zero–knowledge proof of knowledge for a language L defined by relation $\mathcal{R}$ consists of two interactive PPT algorithms $\mathcal{P}$ and $\mathcal{V}$ such that the following conditions hold:

1. **Completeness** For all $(x, w) \in \mathcal{R}$, $\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x) \rangle = 1] = 1$.
2. **Soundness** For all $x \notin \mathsf{L}$, for all adversaries $\mathcal{A}$, the probability $\Pr[\langle \mathcal{A}(x), \mathcal{V}(x) \rangle = 1]$ is negligible. Perfect soundness is achieved if this probability is 0.
3. **Zero–Knowledge** For all PPT adversaries $\mathcal{A}$, there exists a probabilistic PPT algorithm S such that for all $(x, w) \in R$, then the distributions of $\langle \mathcal{P}(x, w), \mathcal{A}(x) \rangle$ and $S(x)$ are computationally indistinguishable. Perfect zero–knowledge is satisfied if the two distributions are identically distributed.
4. **Knowledge Soundness with error** κ There exists a PPT E such that for every adversary $\mathcal{A}$ that outputs x such that $\Pr[\langle \mathcal{A}, \mathcal{V}(x) \rangle = 1] = \epsilon(|x|) > \kappa(|x|)$, then $E^{\mathcal{A}()}(x)$ outputs w such that $(x, w) \in \mathcal{R}$ with probability $\epsilon(|x|) - \kappa(|x|)$.

2.6.1 Non–Interactive Zero–Knowledge Proofs

In a *non–interactive* zero–knowledge proof, the prover does not interact with the verifier. Instead, all parties have access to a trusted *common reference string* (CRS).

Definition 2.9 (Non–Interactive Proof Systems)**.** A set of algorithms $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ constitute a non-interactive (NI) proof system for an efficient relation $\mathcal{R}$ with associated language $L_{\mathcal{R}}$ if completeness and soundness are satisfied. A NI proof system is extractable if, in addition, the extractability property is satisfied. An NI proof system is zero–knowledge (NIZK) if the zero–knowledge property is satisfied. A NIZK proof system that is also extractable constitutes a non-interactive zero–knowledge proof of knowledge (NIZKPoK). An NI proof system is witness–indistinguishable (NIWI) if the witness–indistinguishability property is satisfied. A NIWI proof system that is also extractable constitutes a non-interactive zero–knowledge proof of knowledge (NIWIPoK).

1. **Completeness**. For all $\sigma_{\mathsf{crs}} \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\tau)$ and $(x, w) \in \mathcal{R}$, $\mathcal{V}(\sigma_{\mathsf{crs}}, x, \pi) = 1$ for all proofs $\pi \leftarrow\!\!\$\, \mathcal{P}(\sigma_{\mathsf{crs}}, x, w)$.

2. **Soundness**. For all PPT adversaries $\mathcal{A}$, and for $\sigma_{\mathsf{crs}} \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\tau)$, the probability that $\mathcal{A}(\sigma_{\mathsf{crs}})$ outputs (x, π) such that $x \notin L$ but $\mathcal{V}(\sigma_{\mathsf{crs}}, x, \pi) = 1$ is negligible. Perfect soundness is achieved when this probability is 0.

3. **Extractability**. There exists polynomial–time extractor algorithms $E = (E_1, E_2)$, with $E_1(1^\tau)$ outputting $(\sigma_{\mathsf{ext}}, \tau_e)$, and $E_2(\sigma_{\mathsf{ext}}, \tau_e, x, \pi)$ outputting w, such that:

   (a) $E_1(1^\tau)$ outputs σ_{ext} that is indistinguishable from σ_{crs} output by $\mathsf{CRSSetup}(1^\tau)$.

   (b) for all PPT adversaries $\mathcal{A}$, the probability that $\mathcal{A}(\sigma_{\mathsf{ext}}, \tau_e)$ (where $(\sigma_{\mathsf{ext}}, \tau_e) \leftarrow\!\!\$\, E_1(1^\tau)$) outputs (x, π) such that $\mathcal{V}(\sigma_{\mathsf{crs}}, x, \pi) = 1$ and $(x, E_2(\sigma_{\mathsf{ext}}, \tau_e, x, \pi) \notin \mathcal{R}$, is negligible.

   Perfect extractability is achieved if this probability is 0, and σ_{ext} is distributed identically to σ_{crs}.

4. **Zero–Knowledge**. There exists a polynomial–time simulator algorithm $S = (S_1, S_2)$, with $S_1(1^\tau)$ outputting $(\sigma_{\mathsf{sim}}, \tau_s)$, and $S_2(\sigma_{\mathsf{sim}}, \tau_s, x)$ outputting π_s, such that for all $(x, w) \in \mathcal{R}$ and PPT adversaries $\mathcal{A}$, the following two interactions are indistinguishable:

- We give the adversary $\mathcal{A}$ the CRS $\sigma_{\mathsf{crs}} \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\tau)$ and oracle access to $\mathcal{P}(\sigma_{\mathsf{crs}}, \cdot, \cdot)$ (where $\mathcal{P}$ will output $\perp$ on input (x, w) such that $(x, w) \notin \mathcal{R}$).
- We compute $(\sigma_{\mathsf{sim}}, \tau_s) \leftarrow\!\!\$\, S_1(1^\tau)$ and give the adversary $\mathcal{A}$ the simulated CRS σ_{sim} and oracle access to $S_2(\sigma_{\mathsf{sim}}, \tau_s, \cdot, \cdot)$, where, on input (x, w), S outputs $S_2(\sigma_{\mathsf{sim}}, \tau_s, x)$ if $(x, w) \in \mathcal{R}$ and $\perp$ otherwise.

Perfect zero–knowledge is achieved if for all $(x, w) \in \mathcal{R}$, these interactions are distributed identically.

5. **Witness–Indistinguishability**. For all (x, w_1, w_2) such that $(x, w_1), (x, w_2) \in \mathcal{R}$, the tuple $(\sigma_{\mathsf{crs}}, \pi_1)$ is indistinguishable from $(\sigma_{\mathsf{crs}}, \pi_2)$ where $\sigma_{\mathsf{crs}} \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\tau)$, and for $i \in \{1, 2\}$, $\pi_i \leftarrow\!\!\$\, \mathcal{P}(\sigma_{\mathsf{crs}}, x, w_i)$. Perfect witness indistinguishability is achieved when these two distributions are identical.

Simulation Sound Extractability The *simulation soundness* requirement [52, 116] ensures that soundness holds, even when the adversary is able to see simulated proofs as in the zero–knowledge property. This was strengthened in [75] to ensure that extractability holds, even when the adversary is able to see simulated proofs. All proofs that were not simulated should be extractable.

Definition 2.10 (Simulation Sound Extractability)**.** Let $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V})$ be a NIZKPoK system for an efficient relation $\mathcal{R}$, with a simulator (S_1, S_2) and an extractor (E_1, E_2). Let SE_1 be an algorithm that, on input 1^τ, outputs $(\sigma_{\mathsf{ext}}, \tau_s, \tau_e)$ such that $(\sigma_{\mathsf{ext}}, \tau_s)$ is distributed identically to the output of S_1. Consider the following game with the adversary $\mathcal{A}$:

- $(\sigma_{\mathsf{crs}}, \tau_s, \tau_e) \leftarrow\!\!\$\, SE_1(1^\tau)$
- $(x, \pi) \leftarrow\!\!\$\, \mathcal{A}^{S_2(\sigma_{\mathsf{crs}}, \tau_s, \cdot)}(\sigma_{\mathsf{crs}}, \tau_e)$
- $w \leftarrow E_2(\sigma_{\mathsf{crs}}, \tau_e, x, \pi)$.

The proof system satisfies simulation-sound extractability if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that the probability that $\mathcal{V}(\sigma_{\mathsf{crs}}, x, \pi) = 1$ and $(x, \pi) \notin Q$ (where Q is the set of queried statements and their responses) but either $(x, w) \notin \mathcal{R}$ or $w = \perp$ is at most $\mathsf{negl}(\tau)$.

2.6.2 Fiat–Shamir Transform

The *Fiat–Shamir transform* [63] allows for interactive proofs to be transformed into non-interactive ones in the random oracle model.

The *honest–verifier* zero–knowledge property is a weakening of standard zero–knowledge such that the verifier now performs their side of the protocol honestly. An interactive, constant round, honest–verifier zero–knowledge proof of knowledge Π where the verifier is *public coin*, i.e. its messages are fresh, random coins, and can be transformed into a non-interactive proof Π_{FS} as follows.

- The verifier chooses a hash function.
- The prover executes the protocol, replacing messages with the verifier by the hash of the transcript so far.
- The verifier checks that the transcript would have been accepted, and is consistent with the hash function.

If we model the hash function in the random oracle model, then the security of the interactive proof system Π implies the security of Π_{FS} [4, 62, 113]. Proofs can be simulated by programming the random oracle. Extraction occurs by rewinding and programming the random oracle model, due to the *Forking Lemma* [114]. This lemma roughly says that if an adversary has a non–negligible success probability in proving a statement then, after rewinding to a crucial random oracle query and changing the response, the adversary will generate a second successful proof with non-negligible probability.

2.6.3 Proofs of Discrete Logarithms

We follow the notation defined in [35] when referring to zero–knowledge proofs of knowledge of discrete logarithms. For example $\mathsf{PK}\{(a,b,c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$ denotes a zero–knowledge proof of knowledge of integers a, b and c such that $y = g^a h^b$ and $\tilde{y} = \tilde{g}^a \tilde{h}^c$ hold. Given a protocol in this notation, it is straightforward to derive an actual protocol implementing the proof [35]. Indeed, the computational complexities of the proof protocol can be easily derived from this notation: for each term $y = g^a h^b$, the prover and the

verifier have to perform an equivalent computation, and to transmit one group element and one response value for each exponent.

An SPK denotes a *signature proof of knowledge* [41, 46] which attests that the signer of a message had knowledge of a witness to a particular relation. A formal definition of signature proofs of knowledge is given in [46]. This can be seen as a non-interactive transformation of a proof PK with respect to a message, e.g., using the Fiat–Shamir transform [63] in the random oracle model. Using the Fiat–Shamir transform, the witness can be extracted from these proofs by rewinding the prover and programming the random oracle. We require the proof system to be *simulation-sound* and *zero–knowledge*.

Alternatively, these proofs can be extended to be *online-extractable* [64, 110]. This ensures that the witness can be extracted without rewinding, to avoid potential exponential time blow ups in security reductions. This is achievable by verifiably encrypting the witness to a public key defined in the common reference string. Clearly this requires a trusted common reference string. A practical instantiation is given by Camenisch and Shoup [40] using *Paillier* encryption, secure under the DCR assumption [109]. We underline the values that we need to be online-extractable in all proofs throughout this thesis.

2.7 Group Signature Schemes

Group signature schemes allow users to sign messages on behalf of the group in an anonymous way. That is, a verifier of a group signature is assured that it was signed by a valid member of the group, but does not learn anything about the identity of the signer, or even whether two signatures stem from the same user.

Group signature schemes were first introduced by Chaum and van Heyst [49]. A formal security model was given for the *static* setting, when all users join the scheme at the beginning, in [12]. A security model was given for the *dynamic* setting, where users can join the group at any time, in [14]. We will use this dynamic setting predominately in this thesis, and so we follow this model in this chapter. The dynamic model has three main differences to the static model. Firstly, there is an interactive join protocol, which ensures that users can join at any time and the group manager does not discover the user's secret key. Secondly, the group manager is split into two entities: a group manager who can join

users, and an opener who can de-anonymise signatures. Finally, in the dynamic model the opener produces a proof that it has correctly traced a signature, which can be publicly verified.

Weakened models for group signature schemes have also been proposed to allow for more efficient schemes. In [21], CPA anonymity was defined for group signatures. Analogously to CPA security for encryption schemes, in the CPA anonymity security game there is no opening oracle. Also, in [23], selfless anonymity was introduced where signatures can be de-anonymised if the signer's secret key is leaked. This can be seen as a group signature scheme without forward anonymity.

In [118], an additional security requirement opening soundness was introduced, which ensures that corrupted users cannot claim that the signature of an honest user traces back to them. In [108], randomness exposure resilience of group signatures was defined. It was shown that it is impossible to achieve full anonymity, whilst also achieving randomness exposure resilience. However, randomness exposure resilience can be achieved for schemes with selfless anonymity. In [55, 56], the property of leak-freeness was introduced, which ensures that a signer is not able to convince others that they were responsible for a signature. A security model was also given in [24] for the *fully dynamic* setting, where users are able to join and leave the group at any time, allowing for revocation.

Group signature schemes that are secure without assuming the random oracle model were proposed in [8, 12, 25, 76]. Some group signature schemes have been designed to prioritise very short signatures [17, 21]. *Post-quantum* secure group signature schemes [87, 88, 89, 92, 93] have also been proposed, with the first scheme from lattice assumptions introduced in [74].

2.7.1 Security Model for Dynamic Group Signature Schemes

We first introduce the syntax of dynamic group signature schemes and then present the desirable security and privacy properties for such schemes.

The following entities are involved: an *issuer* $\mathcal{I}$, a set of users $\mathcal{U} = \{uid_i\}$, and an *opener* $\mathcal{O}$. The issuer $\mathcal{I}$ is the central entity that allows users to join the group. Once joined, a

user can then sign on behalf of the group in a pseudonymous way. That is the validity of a signature with respect to the *group's public key* can be publicly verified without revealing any information about the particular user that created the signature. An opener $\mathcal{O}$ can de-anonymise all signatures, outputting an identifier and a proof of this claim.

A group signature scheme in the dynamic setting consists of the following algorithms:

- $\mathsf{GSetup}(1^\tau)$: takes as input the security parameter, and outputs the public parameters $\mathsf{param}_{\mathsf{GS}}$.
- $\mathsf{GKG}(\mathsf{param}_{\mathsf{GS}})$: takes as input the public parameters $\mathsf{param}_{\mathsf{GS}}$, and outputs the group public key gpk, the issuing secret key isk, and the opening secret key osk.
- $\mathsf{GUKG}(1^\tau)$: ran by the user with identifier uid on input the security parameter, and outputs the user's public key $\mathbf{upk}[uid]$ and private key $\mathbf{usk}[uid]$.
- $\langle \mathsf{GJoin}(gpk, \mathbf{usk}[uid], \mathbf{upk}[uid]), \mathsf{GIssue}(isk, gpk, uid, \mathbf{upk}[uid]) \rangle$: a user uid joins the group by engaging in an interactive protocol with the Issuer $\mathcal{I}$. The user uid and Issuer $\mathcal{I}$ perform GJoin and GIssue respectively. These are input a state and an incoming message, and output an updated state, an outgoing message, and a decision, either cont, accept, or reject. The initial input to GJoin is the group public key gpk, and the user uid's public and private key, whereas the initial input to GIssue is the issuer secret key isk, and the user uid's public key. If the issuer accepts, GJoin has a private output of $\mathbf{gsk}[uid]$, and GIssue has output $\mathbf{reg}[uid]$.
- $\mathsf{GSign}(m, \mathbf{gsk}[uid], gpk)$: performed by the user with identifier uid, takes as input a message m, the user's secret key $\mathbf{gsk}[uid]$, and the group public key gpk, and outputs a signature Ω.
- $\mathsf{GVerify}(m, \Omega, gpk)$: outputs 1 if Ω is a valid signature on m under the group public key gpk, and 0 otherwise.
- $\mathsf{GOpen}(m, \Omega, \mathbf{reg}, osk, gpk)$: takes as input a message m, a signature Ω, the registration table $\mathbf{reg}$, the opening secret key osk, and a group public key gpk. Outputs (uid, λ) where if $uid \neq \perp$ then uid is the author of signature Ω and λ is a proof of this claim, and otherwise no group member has been found to author Ω.
- $\mathsf{GJudge}(uid, \mathbf{upk}[uid], m, \Omega, \lambda, gpk)$: takes as input a user identifier uid, the public key $\mathbf{upk}[uid]$ of the user uid, a message m, a valid signature Ω of m, a proof-string

λ and the group public key gpk. It checks that λ is a valid proof that user uid produced Ω, and if so outputs 1 but otherwise outputs 0.

In Figure 2.1, we provide the oracles used in the security requirements. We now provide a high–level description. The joining of users is modelled with the ADDU, SNDU and SNDI oracles, depending on the corruption setting. When the issuer is corrupted, the adversary can already create corrupted users, and can create honest users with the SNDU oracle which runs the join protocol on behalf of this honest user. When the issuer is honest, the adversary can create honest users with the ADDU oracle which runs both the user's and issuer's sides of the protocol, and corrupted users with the SNDI oracle, which runs the issue protocol on behalf of the honest issuer. As the issuer outputs the user's registration after a successfully completed join protocol, after the SNDI and ADDU oracles this registration is stored in **reg**. The oracle WREG allows the adversary to control **reg** if the issuer is corrupted, and the RREG oracle allows the adversary to read **reg** if the adversary is not corrupted. The USK oracle can be used to reveal the secret key of an honest user. The join protocol is modelled as follows: CLS.Join and CLS.Issue both take as input the state $\mathsf{st}^{uid}_{\mathsf{Join}}$ or $\mathsf{st}^{uid}_{\mathsf{Issue}}$, which includes initial inputs and values stored from the previous stage of the protocol, and an incoming message. They output a new state, an outgoing message, and $\mathbf{dec}^{uid}$ which is set to accept, reject or cont and indicates respectively that the protocol has completed successfully, the protocol has completed in failure, or the protocol has not yet completed.

ADDU (join of honest user and honest issuer) Creates a new honest user for uid and internally runs a join protocol between the honest user and honest issuer. At the end, the honest user's secret key $\mathbf{gsk}[uid]$ is generated and from then on signing queries for uid will be allowed.

SNDU (join of honest user and corrupt issuer) Creates a new honest user for uid and runs the join protocol on behalf of uid with the corrupt issuer. If the join session completes, the oracle will store the user's secret key $\mathbf{gsk}[uid]$.

SNDI (join of corrupt user and honest issuer) Runs the join protocol on behalf of the honest issuer with corrupt users. For joins of honest users, the ADDU oracle must be used.

USK Allows an adversary to obtain the secret key of an honest user.

RREG When the issuer is honest, allows an adversary to read **reg**.

WREG When the issuer is corrupt, allows an adversary to write to **reg**.

SIGN Returns signatures for honest users that have successfully joined (via ADDU or SNDU, depending on the corruption setting).

OPEN Allows an adversary to open signatures with GOpen.

All oracles have access to the following records maintained as global state:

HUL List of *uid*s of honest users, initially set to $\emptyset$. New honest users can be added by queries to the ADDU oracle (when the issuer is honest) or SNDU oracle (when the issuer is corrupt).

CUL List of corrupt users that have requested to join the group. Initially set to $\emptyset$, new corrupt users can be added through the SNDI oracle if the issuer is honest. If the issuer is corrupt, we do not keep track of corrupt users.

The security requirements for group signature schemes in the dynamic setting are *correctness*, *anonymity*, *traceability* and *non–frameability*.

Correctness. The correctness requirement is given by the game in Figure 2.2. Given a user is honestly registered to the scheme, and GSign is performed honestly, then the signature output should verify correctly, and correctly open to reveal the correct author of the signature, along with a valid proof of this.

A group signature scheme Π satisfies correctness if for all adversaries $\mathcal{A}$, $\Pr[\mathbf{Exp}^{corr}_{\mathcal{A},\Pi}(\tau) = 1] = 0$.

Anonymity. The anonymity requirement is given by the game in Figure 2.2, and ensures that a user's signature cannot be de-anonymised or linked by user. In the security game the adversary has corrupted the issuer but not the opener, as otherwise they could trivially open signatures. They choose two honest users and a message, and in response are given a signature authored by one of these authors and must guess which. They can create honest users using the SNDU oracle and corrupt all honest users, including the two users challenged by the adversary, with the USK oracle. As they are assumed to have corrupted

ADDU(uid):

if $uid \in \mathsf{HUL} \cup \mathsf{CUL}$ **return** $\perp$
$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp$
$\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, (\mathbf{upk}[uid], \mathbf{usk}[uid]) \leftarrow\!\!\$\ \mathsf{GUKG}(1^\tau)$
$\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk, \mathbf{usk}[uid], \mathbf{upk}[uid])$
$\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk, uid, \mathbf{upk}[uid])$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{GJoin}(\mathsf{st}^{uid}_{\mathsf{Join}}, \perp)$
while $\mathbf{dec}^{uid} = \mathsf{cont}$
  $(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Join}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{GIssue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Issue}})$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{GJoin}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Join}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$
  $\mathbf{reg}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Issue}}, \mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $\mathbf{upk}[uid]$

SNDU(uid, M_{in}):

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}$
  $(\mathbf{upk}[uid], \mathbf{usk}[uid]) \leftarrow\!\!\$\ \mathsf{GUKG}(1^\tau)$
  $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if $\mathsf{st}^{uid}_{\mathsf{Join}}$ undefined
  $\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk, \mathbf{usk}[uid], \mathbf{upk}[uid])$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{GJoin}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

SNDI($uid, M_{\mathsf{in}}, upk$):

if $uid \in \mathsf{HUL}$ **return** $\perp$
if $uid \notin \mathsf{CUL}$
  $\mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{uid\}$
  $\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, \mathbf{upk}[uid] \leftarrow upk$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if undefined $\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk, uid, \mathbf{upk}[uid])$
$(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{GIssue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{in}}, \mathbf{dec}^{uid})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{reg}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Issue}}$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

USK(uid):

return $(\mathbf{gsk}[uid], \mathbf{usk}[uid])$

RREG(uid):

return $\mathbf{reg}[uid]$

WREG(uid, reg):

$\mathbf{reg}[uid] \leftarrow reg$

SIGN(m, uid):

if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** $\perp$
return $\mathsf{GSign}(m, \mathbf{gsk}[uid], gpk)$

OPEN(m, Ω):

if $(m, \Omega) = (m^*, \Omega^*)$ **return** $\perp$
return $\mathsf{GOpen}(m, \Omega, \mathbf{reg}, osk, gpk)$

Figure 2.1: Oracles used in the dynamic group signature security model

the issuer, they have access to WREG. They have access to the OPEN oracle, however they cannot query the challenge signature, to avoid trivial wins.

Definition 2.11 (Anonymity)**.** A group signature scheme Π satisfies anonymity if for all polynomial–time adversaries $\mathcal{A}$, the advantage $|\Pr[\mathbf{Exp}^{anon-0}_{\mathcal{A},\Pi}(\tau) = 1] - \Pr[\mathbf{Exp}^{anon-1}_{\mathcal{A},\Pi}(\tau) = 1]|$ is negligible in τ.

Traceability. The traceability requirement is given by the game in Figure 2.2. This requirement ensures that all valid signatures open to reveal a valid user, along with a valid proof under GJudge, when the issuer is honest. In the security game the adversary has corrupted the opener but not the issuer, as otherwise they could simply create users without updating **reg** and so produce valid signatures that would not open correctly. They

are given the SNDI and ADDU oracles to create honest and corrupt users respectively, and the RREG oracle to read **reg**. The USK oracle allows them to reveal the private keys of honest users. The adversary must output a valid signature that does not open to reveal a user along with a valid proof.

Definition 2.12 (Traceability)**.** A group signature scheme Π satisfies traceability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\Pi}(\tau) = 1]$ is negligible in τ.

Non–frameability. The non–frameability requirement is given by the game in Figure 2.2. This requirement ensures an adversary cannot impersonate an honest user, by forging a signature that opens to their user identifier. In the security game the adversary has corrupted the opener and the issuer, but not all users. They have access to the SNDU oracle, which allows them to create honest users, and the USK oracle, which allows them to obtain these users' private keys. As they have corrupted the issuer they have access to the WREG oracle which allows them to control **reg**. They can obtain signatures of honest users with the SIGN oracle. Without using the signing oracle, the adversary must output a valid signature, the identifier of an honest user that has not been revealed with USK, and a proof that is valid under GJudge with respect to this signature and user identifier.

Definition 2.13 (Non–frameability)**.** A group signature scheme Π satisfies non–frameability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{non-frame}_{\mathcal{A},\Pi}(\tau) = 1]$ is negligible in τ.

2.7.2 XS Group Signatures

We now present the XS group signature scheme which will be used in Chapter 3. The XS scheme [53] satisfies the security requirements for dynamic group signatures [14], under the q-SDH [19] assumption, the XDH assumption [10, 21], and in the random oracle model [13].

As the XS scheme is given in the type–2 setting, it relies on the *XDH* assumption.

Definition 2.14 (XDH Assumption)**.** The DDH assumption holds in $\mathbb{G}_1$ even when there exists a homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$.

We provide the XS scheme in Figures 2.3 and 2.4. The public parameters of the XS scheme are the type–2 bilinear group: $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2, \Psi)$.

Experiment: $\mathbf{Exp}^{corr}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param}_{\mathsf{GS}} \leftarrow\!\!\$\ \mathsf{GSetup}(1^\tau), (gpk, isk, osk) \leftarrow\!\!\$\ \mathsf{GKG}(\mathsf{param}_{\mathsf{GS}}), \mathsf{CUL}, \mathsf{HUL} \leftarrow \emptyset$
$(uid, m) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{ADDU},\mathsf{RREG}}(gpk)$
if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** 0
$\Omega \leftarrow\!\!\$\ \mathsf{GSign}(m, \mathbf{gsk}[uid], gpk)$, **if** $\mathsf{GVerify}(m, \Omega, gpk) = 0$ **return** 1
$(uid', \lambda) \leftarrow\!\!\$\ \mathsf{GOpen}(m, \Omega, \mathbf{reg}, osk, gpk)$, **if** $uid \neq uid'$ **return** 1
if $\mathsf{GJudge}(uid, \mathbf{upk}[uid], m, \Omega, \lambda, gpk) = 0$ **return** 1 **else return** 0

Experiment: $\mathbf{Exp}^{anon-b}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param}_{\mathsf{GS}} \leftarrow\!\!\$\ \mathsf{GSetup}(1^\tau), (gpk, isk, osk) \leftarrow\!\!\$\ \mathsf{GKG}(\mathsf{param}_{\mathsf{GS}}), \mathsf{CUL}, \mathsf{HUL} \leftarrow \emptyset$
$(\mathsf{st}, uid_0, uid_1, m^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{OPEN},\mathsf{SNDU},\mathsf{WREG},\mathsf{USK}}(\mathsf{choose}, gpk, isk)$
if $uid_0, uid_1 \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0], \mathbf{gsk}[uid_1] = \perp$ **return** 0
$\Omega^* \leftarrow\!\!\$\ \mathsf{GSign}(m^*, \mathbf{gsk}[uid_b], gpk), b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{OPEN},\mathsf{SNDU},\mathsf{WREG},\mathsf{USK}}(\mathsf{guess}, \mathsf{st}, \Omega^*)$
return b^*

Experiment: $\mathbf{Exp}^{trace}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param}_{\mathsf{GS}} \leftarrow\!\!\$\ \mathsf{GSetup}(1^\tau), (gpk, isk, osk) \leftarrow\!\!\$\ \mathsf{GKG}(\mathsf{param}_{\mathsf{GS}}), \mathsf{CUL}, \mathsf{HUL} \leftarrow \emptyset$
$(m, \Omega) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDI},\mathsf{ADDU},\mathsf{RREG},\mathsf{USK}}(gpk, osk)$
if $\mathsf{GVerify}(m, \Omega, gpk) = 0$ **return** 0
$(uid, \lambda) \leftarrow\!\!\$\ \mathsf{GOpen}(m, \Omega, \mathbf{reg}, osk, gpk)$
if $uid = \perp$ or $\mathsf{GJudge}(uid, \mathbf{upk}[uid], m, \Omega, \lambda, gpk) = 0$ **return** 1 **else return** 0

Experiment: $\mathbf{Exp}^{non-frame}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param}_{\mathsf{GS}} \leftarrow\!\!\$\ \mathsf{GSetup}(1^\tau), (gpk, isk, osk) \leftarrow\!\!\$\ \mathsf{GKG}(\mathsf{param}_{\mathsf{GS}}), \mathsf{CUL}, \mathsf{HUL} \leftarrow \emptyset$
$(m, \Omega, uid, \lambda) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU},\mathsf{WREG},\mathsf{SIGN},\mathsf{USK}}(gpk, isk, osk)$
if $\mathsf{GVerify}(m, \Omega, gpk) = 0$ **return** 0
If the following conditions hold **return** 1 **else return** 0
1. $uid \in \mathsf{HUL}$ and $\mathbf{gsk}[uid] \neq \perp$
2. $\mathsf{GJudge}(uid, \mathbf{upk}[uid], m, \Omega, \lambda, gpk) = 1$
3. USK not queried uid and SIGN oracle not queried (m, uid)

Figure 2.2: Experiments capturing the correctness, anonymity, traceability, and non–frameability security requirements for dynamic group signature schemes

2.7.3 Variants of group signature schemes

Since the introduction of group signatures, several variants have also been proposed for different applications. Many seek to reduce the power of the opener, which can de-anonymise all signatures and so is potentially a privacy bottleneck. An overview is given in [99].

Ring signatures [115] do not involve openers or issuers. Users generate their own secret and public keys and can sign on behalf of a ring of users including themselves. The signature guarantees that the message originates from a member of this ring, without revealing the

$\mathsf{XSKeyGen}(\mathsf{param})$

$\xi_1, \xi_2 \leftarrow_\$ \mathbb{Z}_p; K \leftarrow_\$ \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}, H \leftarrow K^{\xi_1}, G \leftarrow K^{\xi_2}, \gamma \leftarrow_\$ \mathbb{Z}_p, W \leftarrow G_2^{\gamma}$
return $gpk = (G_1, K, H, G, G_2, W), isk = \gamma, osk = (\xi_1, \xi_2)$

$\mathsf{XSUKG}(1^\tau)$

return $(upk, usk) \leftarrow_\$ \mathsf{SigKeyGen}(1^\tau)$

$\mathsf{XSSign}(m, (Z, x, y), gpk)$

$\rho_1, \rho_2 \leftarrow_\$ \mathbb{Z}_p, T_1 \leftarrow K^{\rho_1}, T_2 \leftarrow ZH^{\rho_1}, T_3 \leftarrow K^{\rho_2}, T_4 \leftarrow ZG^{\rho_2}, z \leftarrow x\rho_1 + y$
$\pi \leftarrow_\$ \mathsf{SPK}\{(x, z, \rho_1, \rho_2) : T_1 = K^{\rho_1} \wedge T_3 = K^{\rho_2} \wedge T_4 T_2^{-1} = G^{\rho_2} H^{-\rho_1} \wedge$
$e(T_2, W)e(T_2, G_2)^x = e(G_1, G_2)e(H, W)^{\rho_1} e(H, G_2)^z\}(m)$
return $\Omega = (T_1, T_2, T_3, T_4, \pi)$

$\mathsf{XSVerify}(m, \Omega, gpk)$

return 1 **if** π holds for T_1, T_2, T_3, T_4 **else return** 0

$\mathsf{XSOpen}(m, \Omega, osk, gpk)$

if $\mathsf{XSVerify}(m, \Omega, gpk) = 0$ **return** $\perp$
$Z \leftarrow T_2 T_1^{-\xi_1}$, **if** $\exists uid$ s.t $\mathbf{reg}[uid] = (upk, Z, ., S)$ and $\mathsf{Verify}(upk, Z, S) = 1$ $uid^* \leftarrow uid$
$\lambda = (Z, \mathsf{SPK}\{\xi_1 : Z = T_2 T_1^{-\xi_1} \wedge H = K^{\xi_1}\})$
return (uid^*, λ)

$\mathsf{XSJudge}(uid, upk, m, \Omega, \lambda = (Z, \pi_{\mathsf{jg}}), gpk)$

If the following hold **return** 1 **else return** 0
1.$\mathsf{XSVerify}(m, \Omega, gpk) = 1$
2.π_{jg} is valid in relation to Z, Ω, gpk
3.$\mathbf{reg}[uid] = (upk, Z, ., S)$ and $\mathsf{Verify}(upk, Z, S) = 1$

Figure 2.3: The XS group signature scheme

identity of the signer. *Linkable ring signatures* [94] allow signatures with the same author to be linked, while signatures still do not reveal the identity of the signer.

Direct anonymous attestation (DAA) schemes [29] also do not have a trusted opener, but instead have user–controlled linkability. Users now sign basenames as well as messages. Signatures with the same basename and signer can be publicly linked together, and otherwise are unlinkable. Therefore, the user re-uses the same basename whenever they wants to be linkable. In DAA the user is split into two entities: a host, which is assumed to have greater computational power, and a *trusted platform module* (TPM). Enhanced privacy ID (EPID) schemes [28] were introduced to allow for revocation in DAA.

$\mathsf{XSJoin}(gpk, \mathbf{upk}[uid], \mathbf{usk}[uid])$	$\rightleftharpoons$	$\mathsf{XSIssue}(isk, gpk, uid, \mathbf{upk}[uid])$
$y \leftarrow_\$ \mathbb{Z}_p^*, C \leftarrow H^y,$		
$\pi_{\mathsf{join}} \leftarrow_\$ \mathsf{SPK}\{y : C = H^y\}$		
	$\xrightarrow{C, \pi_{\mathsf{join}}}$	
		Verifies $C \in \mathbb{G}_1$, checks π_{join},
		$x \leftarrow_\$ \mathbb{Z}_p^*, Z \leftarrow (G_1 C)^{1/(isk+x)},$
		$E \leftarrow e(G_1 C, G_2)/e(Z, W),$
		$D \leftarrow e(Z, G_2)$
		$\pi_{\mathsf{iss}} \leftarrow_\$ \mathsf{SPK}\{x : D^x = E\}$
	$\xleftarrow{Z, \pi_{\mathsf{iss}}}$	
$E \leftarrow e(G_1 C, G_2)/e(Z, W)$		
$D \leftarrow e(Z, G_2)$		
Verifies $Z \in \mathbb{G}_1$, checks π_{iss}		
$S \leftarrow_\$ \mathsf{Sign}(\mathbf{usk}[uid], Z)$		
	$\xrightarrow{S}$	
		Checks $\mathsf{Verify}(\mathbf{upk}[uid], Z, S) = 1$
		$\mathbf{reg}[uid] \leftarrow (\mathbf{upk}[uid], Z, x, S)$
	$\xleftarrow{x}$	
Checks $e(Z, G_2)^x e(Z, W) e(C, G_2)^{-1} =$		
$e(G_1, G_2)$		
$\mathbf{gsk}[uid] \leftarrow (Z, x, y)$		

Figure 2.4: The $\langle \mathsf{XSJoin}, \mathsf{XSIssue} \rangle$ protocol of the XS group signature scheme

Dynamic group signatures with distributed traceability were formalised in [72]. These allow a set of managers to open signatures in a distributed way, reducing the power of the opener.

Forward secure group signatures [121, 104] provide security even after a user's signing key is leaked, by updating the users' secret keys every time interval. This goes further than the forward anonymity mentioned earlier, which only ensures that anonymity is preserved after a user's secret key is leaked.

Group signatures with deniability were introduced in [81]. These allow the opener to prove that a user was not the signer of a signature, without revealing the actual signer.

In group signatures with *controlled linkability* [79, 80, 120], signatures are unlinkable except to a dedicated linking authority with a linking secret key. On input two signatures this linking authority decides whether they stem from the same user or not.

Another related concept is *traceable* group signatures [83] where a dedicated entity can generate a tracing trapdoor for each user which allows this user's signatures to be traced. They also allow group members to prove they authored a signature.

Accountable tracing signatures [84] also reduce the power of the opener, by allowing the opener to choose which users' signatures they can open and prove they did not have the ability to open a particular user's signatures.

In [85], they avoid the need for an opener all together, by allowing users to prove or deny authorship of a signature, as well as prove that two signatures authored by them are linked. The opener can also prove that two signatures originate from the same user without revealing user identities.

In [117], the power of the opener is reduced by introducing another entity, the admitter. They have the power to specify messages, so that only signatures on those messages should be able to be opened.

Different approaches have been proposed to allow for *revocation* in group signatures. One approach is for users to prove when they sign that they are not included in a revocation list [26, 34, 38, 57, 90, 91, 103, 107, 124]. Another method is to regularly update the group public key, and users' secret keys, so that revoked users are no longer able to sign [21, 121]. In verifier local revocation [23, 27], revocation information is sent to the verifiers who can then determine whether a revoked user authored a signature. Group signature schemes with verifier local revocation, such that the revocation is probabilistic have also been proposed [86]. Another type of group signatures with verifier local revocation is group signatures with time bound keys [61, 50]. Users' secret keys are bound to a specific time interval, and after these keys expire they are immediately revoked. The model for fully dynamic group signatures given in [24] allows for revocation capturing the first two of these approaches.

Another variant is group blind signature schemes [96]. Blind signatures [48] allow a message to be signed via an interactive protocol between a entity holding the signing secret key who should not learn the message and an entity holding the message who should not learn the secret key. Group blind signature schemes similarly allow group signatures to be signed in this way.

Democratic group signatures [98] remove the role of the group manager, so that new members join and leave the group via an interactive protocol with all existing group members. There is no longer an opener, and instead all group members hold the tracing trapdoor. Linkable democratic group signatures [100] allow signatures to be linked (but not traced) by entities outside the group.

Mediated group signature schemes [55, 56] introduce an additional central entity, the mediator. The signer identifies themself and submits a partial signature to the mediator, which converts this into a full signature. This means that the mediator can immediately revoke a group member.

2.8 Direct Anonymous Attestation

Direct anonymous attestation was introduced in [29]. A security model was given in [15] for pre–DAA, the case where the TPM and host are merged into one entity. Security requirements for the full DAA model were given in [33].

2.8.1 Pre–DAA Security Model

As in this thesis we always assume the host and TPM are merged, we now present the *pre–DAA* security model [15]. A pre–DAA scheme consists of the following algorithms.

- $\mathsf{DAASetup}(1^\tau)$: takes as input the security parameter τ and outputs the parameters for the scheme param.
- $\mathsf{DAAKeyGen}(\mathsf{param})$: takes as input the parameters param, and outputs the group public key gpk, and the issuing secret key isk.
- $\langle \mathsf{DAAJoin}(gpk), \mathsf{DAAIssue}(isk, gpk) \rangle$: a user uid joins the group by engaging in an interactive protocol with the Issuer. The user with identifier uid and Issuer perform algorithms $\mathsf{DAAJoin}$ and $\mathsf{DAAIssue}$ respectively. These are input a state and an incoming message respectively, and output an updated state, an outgoing message, and a decision, either cont, accept, or reject. The initial input to $\mathsf{DAAJoin}$ is the

group public key, whereas the initial input to DAAIssue is the issuer secret key, isk, and the group public key. If the issuer accepts, DAAJoin has a private output of $\mathbf{gsk}[uid]$, DAAIssue has a private output of $\mathbf{reg}[uid]$.

- DAASign$(bsn, m, \mathbf{gsk}[uid], gpk)$: takes as input a basename bsn, a message m, a user secret key $\mathbf{gsk}[uid]$, and a group public key gpk, and outputs a signature Ω.
- DAAVerify(bsn, m, Ω, gpk): takes as input a basename bsn, a message m, a signature Ω, and a group public key gpk, outputs 1 if Ω is valid, and 0 otherwise.
- DAALink$((bsn_0, m_0, \Omega_0), (bsn_1, m_1, \Omega_1), gpk)$ takes as input two signatures Ω_0, Ω_1 each on a basename and a message, and a group public key gpk. Outputs 1 if the two signatures are linked and 0 otherwise.
- DAAIdentify$_T(\mathcal{T}, gsk)$ outputs 1 if $\mathcal{T}$ corresponds to a valid transcript of a $<$ DAAJoin, DAAIssue $>$ with output gsk, and otherwise 0.
- DAAIdentify$_S(bsn, m, \Omega, gsk)$ outputs 1 if the signature Ω originates from the user secret key gsk, and 0 otherwise.

In Figure 2.5, we provide the oracles used in the pre–DAA security requirements: ADDU, SNDU, SNDI, USK, and SIGN. These are similar to those for dynamic group signatures. As there is no longer an opening functionality, there is no need for the WREG and RREG oracles. The ADDU, SNDU, and SNDI oracles allow for the joining of honest and corrupted users, depending on the corruption setting of the issuer as in group signature schemes. The USK again allows the secret keys of honest users to be revealed, and the SIGN oracle outputs signatures of honest users. There is no need for a linking oracle, because signatures are publicly linkable. As in group signatures, DAAJoin and DAAIssue both take as input the state, and an incoming message. They output a new state, an outgoing message, and $\mathbf{dec}^{uid}$ which again is set to accept, reject or cont.

All oracles have access to the following records maintained as global state:

HUL List of uids of honest users, initially set to $\emptyset$. New honest users can be added by queries to the ADDU oracle (when the issuer is honest) or SNDU oracle (when the issuer is corrupt).

CUL List of corrupt users that have requested to join the group. Initially set to $\emptyset$, new corrupt users can be added through the SNDI oracle if the issuer is honest. If the issuer is corrupt, we do not keep track of corrupt users.

BUL List of users that have been corrupted with the USK oracle. Initially set to $\emptyset$, only honest users that have joined through the SNDU or ADDU oracles can be corrupted in this way.

SL List of (uid, m, bsn) tuples input to the SIGN oracle.

ADDU(uid):

if $uid \in \mathsf{HUL} \cup \mathsf{CUL}$ **return** $\perp$
$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp$
$\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, \mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk)$
$\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk, uid)$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{DAAJoin}(\mathsf{st}^{uid}_{\mathsf{Join}}, \perp)$
while $\mathbf{dec}^{uid} = \mathsf{cont}$
  $(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Join}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{DAAIssue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Issue}})$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{DAAJoin}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Join}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$
  $\mathbf{reg}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Issue}}, \mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $\mathbf{reg}[uid]$

SNDU(uid, M_{in}):

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}$
  $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if $\mathsf{st}^{uid}_{\mathsf{Join}}$ undefined $\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow gpk$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{DAAJoin}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

SNDI(uid, M_{in}):

if $uid \in \mathsf{HUL}$ **return** $\perp$
if $uid \notin \mathsf{CUL}$
  $\mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{uid\}, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if undefined $\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk, uid)$
$(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{DAAIssue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{in}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{reg}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Issue}}$
  return $(M_{\mathsf{out}}, \mathbf{dec}^{uid}, \mathbf{reg}[uid])$
else return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

USK(uid):

if $uid \notin \mathsf{HUL}$ **return** $\perp$
$\mathsf{BUL} \leftarrow \mathsf{BUL} \cup \{uid\}$ **return** $\mathbf{gsk}[uid]$

SIGN(bsn, m, uid):

if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** $\perp$
$\mathsf{SL} \leftarrow \mathsf{SL} \cup \{uid, m, bsn\}$
return $\mathsf{DAASign}(bsn, m, \mathbf{gsk}[uid], gpk)$

Figure 2.5: Oracles in the pre-DAA security model

The security requirements for pre–DAA schemes are *correctness*, *anonymity*, *traceability* and *non–frameability*.

Correctness. In the game given in Figure 2.6 we provide the correctness requirement. This ensures that, given a user is honestly joined to the scheme and DAASign is performed correctly, the signature output will verify correctly. It also ensures that signatures generated honestly using the same user private key and basename will be linked under DAALink,

Experiment: $\mathbf{Exp}^{corr}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\, \mathsf{DAASetup}(1^\tau), (gpk, isk) \leftarrow\!\!\$\, \mathsf{DAAKeyGen}(\mathsf{param}), \mathsf{HUL}, \mathsf{CUL} \leftarrow \emptyset$
$(uid, m_0, m_1, bsn) \leftarrow\!\!\$\, \mathcal{A}^{\mathsf{ADDU}}(gpk)$, **if** $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** 0
$\forall b \in \{0,1\}$ $\Omega_b \leftarrow\!\!\$\, \mathsf{DAASign}(bsn, m_b, \mathbf{gsk}[uid], gpk)$
$\forall b \in \{0,1\}$ **if** $\mathsf{DAAVerify}(bsn, m_b, \Omega_b, gpk) = 0$ **return** 1
if $bsn \neq \perp$ **if** $\mathsf{DAALink}((bsn, m_0, \Omega_0), (bsn, m_1, \Omega_1), gpk) = 0$ **return** 1
if $\mathsf{DAAIdentify}_S(bsn, m_0, \Omega_0, \mathbf{gsk}[uid]) = 0$ **return** 1
Let $\mathcal{T}$ denote the $< \mathsf{DAAJoin}, \mathsf{DAAIssue} >$ transcript for user uid
if $\mathsf{DAAIdentify}_T(\mathcal{T}, \mathbf{gsk}[uid]) = 0$ **return** 1 **else return** 0

Figure 2.6: Experiment capturing the correctness requirement for pre–DAA schemes

and that $\mathsf{DAAIdentify}_S$ and $\mathsf{DAAIdentify}_T$ correctly identify signatures to the user private key and the transcript respectively.

A pre–DAA scheme Π satisfies correctness if for all adversaries $\mathcal{A}$: $\Pr[\mathbf{Exp}^{corr}_{\mathcal{A},\Pi}(\tau) = 1] = 0$.

Anonymity. The anonymity requirement is given by the game in Figure 2.7. This ensures that a user's signatures cannot be de-anonymised, and signatures with different basenames cannot be linked by user. In the security game, the adversary has corrupted the issuer, and chooses two honest users and a message and basename. They are returned with a challenge signature and they must guess which of the two users is the author. They can create honest users using the SNDU oracle and corrupt these with the USK oracle. They can also obtain signatures from honest users with the SIGN oracle. However, they cannot corrupt either of the challenged honest users with the USK, or query one of these users, and the challenge basename to SIGN as otherwise the $\mathsf{DAALink}$ algorithm could be used to trivially win.

Definition 2.15 (Anonymity)**.** A pre–DAA scheme Π satisfies anonymity if for all polynomial–time adversaries $\mathcal{A}$, the advantage $|\Pr[\mathbf{Exp}^{anon-0}_{\mathcal{A},\Pi}(\tau) = 1] - \Pr[\mathbf{Exp}^{anon-1}_{\mathcal{A},\Pi}(\tau) = 1]|$ is negligible in τ.

Traceability. The traceability requirement is given by the game in Figure 2.7. This requirement ensures firstly that all signatures identify under $\mathsf{DAAIdentify}_S$ to a secret key obtained through a $< \mathsf{DAAJoin}, \mathsf{DAAIssue} >$ protocol, and secondly that two signatures on the same basename that identify to the same secret key under $\mathsf{DAAIdentify}_S$ are always linked. The Traceability experiment is made up of two games, which each capture a different property. Both properties must be satisfied to ensure Traceability. The adversary is separated into $\mathcal{A}_1$ and $\mathcal{A}_2$ to emphasize that these are separate games and there is no

Experiment: $\mathbf{Exp}^{anon-b}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{DAASetup}(1^\tau), (gpk, isk) \leftarrow_\$ \mathsf{DAAKeyGen}(\mathsf{param}), \mathsf{HUL}, \mathsf{BUL}, \mathsf{SL} \leftarrow \emptyset$
$(\mathsf{st}, uid_0, uid_1, bsn, m) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{USK},\mathsf{SIGN}}(\mathsf{choose}, gpk, isk)$
if $uid_0, uid_1 \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0], \mathbf{gsk}[uid_1] = \perp$ **return** 0
$\Omega^* \leftarrow_\$ \mathsf{DAASign}(bsn, m, \mathbf{gsk}[uid_b], gpk)$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{USK},\mathsf{SIGN}}(\mathsf{guess}, \mathsf{st}, \Omega^*)$
if $uid_0, uid_1 \in \mathsf{BUL}$ or $(uid_0, \cdot, bsn), (uid_1, \cdot, bsn) \in \mathsf{SL}$ **return** 0
return b^*

Experiment: $\mathbf{Exp}^{trace}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{DAASetup}(1^\tau), (gpk, isk) \leftarrow_\$ \mathsf{DAAKeyGen}(\mathsf{param}), \mathsf{HUL}, \mathsf{CUL}, \mathsf{BUL} \leftarrow \emptyset$
$(\Omega, m, bsn, gsk_1, \cdots, gsk_l) \leftarrow_\$ \mathcal{A}_1^{\mathsf{SNDI}}(gpk)$
Let T denote the set of all transcripts accepted from SNDI queries
If the following conditions hold **return** 1
1. $\mathsf{DAAVerify}(bsn, m, \Omega, gpk) = 1$
2. $\forall \mathcal{T} \in T \quad \exists i \in [l]$ such that $\mathsf{DAAIdentify}_T(\mathcal{T}, gsk_i) = 1$
3. $\forall i \in [l] \quad \mathsf{DAAIdentify}_S(bsn, m, \Omega, gsk_i) = 0$

$(bsn, m_0, m_1, \Omega_0, \Omega_1, gsk) \leftarrow_\$ \mathcal{A}_2(gpk, isk)$ **if** $bsn = \perp$ **return** 0
If the following conditions hold **return** 1 **else return** 0
1. $\forall b \in \{0,1\} \quad \mathsf{DAAVerify}(bsn, m_b, \Omega_b, gpk) = 1$
2. $\forall b \in \{0,1\} \quad \mathsf{DAAIdentify}_S(bsn, m_b, \Omega_b, gsk) = 1$
3. $\mathsf{DAALink}((bsn, m_0, \Omega_0), (bsn, m_1, \Omega_1), gpk) = 0$

Experiment: $\mathbf{Exp}^{non-frame}_{\mathcal{A},\Pi}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{DAASetup}(1^\tau), (gpk, isk) \leftarrow_\$ \mathsf{DAAKeyGen}(\mathsf{param}), \mathsf{HUL}, \mathsf{BUL}, \mathsf{SL} \leftarrow \emptyset$
$(bsn, m, uid, \Omega) \leftarrow_\$ \mathcal{A}_1^{\mathsf{SNDU},\mathsf{USK},\mathsf{SIGN}}(gpk, isk)$
If the following conditions hold **return** 1
1. $\mathsf{DAAVerify}(bsn, m, \Omega, gpk) = 1$
2. $uid \in \mathsf{HUL} \backslash \mathsf{BUL}$ and $(uid, m, bsn) \notin \mathsf{SL}$
3. $\mathsf{DAAIdentify}_S(bsn, m, \Omega, \mathbf{gsk}[uid]) = 1.$

$(bsn_0, m_0, \Omega_0, bsn_1, m_1, \Omega_1, gsk) \leftarrow_\$ \mathcal{A}_2(gpk, isk)$
If one of the following conditions hold **return** 0
1. $\exists b \in \{0,1\}$ such that $\mathsf{DAAVerify}(bsn_b, m_b, \Omega_b, gpk) = 0$
2. $\mathsf{DAALink}((bsn_0, m_0, \Omega_0), (bsn_1, m_1, \Omega_1), gpk) = 0$

If one of the following conditions hold **return** 1 **else return** 0
1. $\mathsf{DAAIdentify}_S(bsn_0, m_0, \Omega_0, gsk) = 1$ and $\mathsf{DAAIdentify}_S(bsn_1, m_1, \Omega_1, gsk) = 0$
2. $bsn_0 \neq bsn_1$ or $bsn_0 = \perp$ or $bsn_1 = \perp$

Figure 2.7: Experiments capturing the anonymity, traceability and non–frameability security requirements for pre–DAA schemes

joint state.

The first game captures that all valid signatures should identify to a secret key of a user that has successfully joined under $\mathsf{DAAIdentify}_S$. The adversary has not corrupted the

issuer as otherwise they could simply create their own unregistered users. They are given the SNDI oracle to create corrupt users, and have no need for other oracles as the game does not distinguish between honest and corrupt users. They then must output a secret key corresponding to every accepted SNDI query under $\mathsf{DAAIdentify}_T$, and a valid signature that does not identify to any of these secret keys under $\mathsf{DAAIdentify}_S$.

The second game captures that signatures tracing to the same secret key and with the same basename should link under DAALink. The adversary has corrupted the issuer, and has no need for oracles as the game does not distinguish between honest and corrupt users. They must output a basename, a user secret key, and two valid signatures on this basename. They win if the two signatures are not linked under DAALink but do identify to the same secret key.

Definition 2.16 (Traceability)**.** A pre–DAA scheme Π satisfies traceability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\Pi}(\tau) = 1]$ is negligible in τ.

Non–frameability. The non–frameability requirement is given by the game in Figure 2.7. This requirement ensures that an adversary cannot impersonate an honest user by forging signatures linking to theirs. This requires firstly that an adversary should not be able to output a valid signature that identifies to the secret key of an honest user under $\mathsf{DAAIdentify}_S$, and secondly that they should not be able to output two valid linked signatures, that either have different basenames or identify under $\mathsf{DAAIdentify}_S$ to different secret keys. The non–frameability experiment is also made up of two games, which each capture a different property. Both properties must be satisfied to ensure Non–frameability. The adversary is separated into $\mathcal{A}_1$ and $\mathcal{A}_2$ to emphasize that these are separate games and there is no joint state.

The first security game captures that an adversary should not be able to output a signature that traces to the secret key of an honest user. The adversary has corrupted the issuer. They are given the SNDU, USK, SIGN oracles to create honest users, reveal their private keys, and obtain signatures from these honest users. They then must output a valid signature that identifies under $\mathsf{DAAIdentify}_S$ to the secret key of an honest user, that was not revealed under USK, or obtained through the SIGN oracle.

The second game captures that signatures tracing to different secret keys or with different basenames should not link under DAALink. The adversary has again corrupted the issuer,

and has no need for oracles as the game does not distinguish between honest and corrupt users. They must output two valid linked signatures and a user secret key. They win if either the basenames of the two signatures are different or only one of the signatures identifies under $\mathsf{DAAIdentify}_S$ to the secret key.

Definition 2.17 (Non–frameability)**.** A pre–DAA scheme Π satisfies non–frameability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{non-frame}_{\mathcal{A},\Pi}(\tau) = 1]$ is negligible in τ.

2.8.2 CDL Direct Anonymous Attestation Scheme

The CDL scheme [33] was proved secure under the state–of–the–art security requirements for full DAA [33], assuming the LSRW [97], Discrete Logarithm (DL), and DDH assumptions. It consists of the algorithms provided in Figure 2.8, and the protocol provided in Figure 2.9. We merge the TPM and host here, as this is the setting considered throughout this thesis. The public parameters of a CDL scheme are the type–3 bilinear group: $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2)$ and a hash function $\mathcal{H}$.

$\mathsf{CDLKeyGen}(\mathsf{param})$

$\alpha, \beta \leftarrow_\$ \mathbb{Z}_p^*, X \leftarrow {G_2}^\alpha, Y \leftarrow {G_2}^\beta$ **return** $gpk = (X, Y), isk = (\alpha, \beta)$

$\mathsf{CDLSign}(bsn, m, (f, (A, B, C, D), gpk)$

$a \leftarrow_\$ \mathbb{Z}_p^*, A' \leftarrow A^a, B' \leftarrow B^a, C \leftarrow C^a, D' \leftarrow D^a$
if $bsn \neq \perp$ $J \leftarrow \mathcal{H}(bsn)^f, \pi \leftarrow_\$ \mathsf{SPK}\{(f) : D' = {B'}^f \wedge J = \mathcal{H}(bsn)^f\}(bsn, m)$
if $bsn = \perp$ $J \leftarrow \perp, \pi \leftarrow_\$ \mathsf{SPK}\{(f) : D' = {B'}^f\}(bsn, m)$
return $\Omega = (A', B', C', D', J, \pi)$

$\mathsf{CDLVerify}(bsn, m, \Omega, gpk)$

Parse $\Omega = (A', B', C', D', J, \pi)$
Verify π with respect to B', D', J, m, bsn
if $A' = 1$ or $J = 1$ **return** 0
if $e(A', Y) \neq e(B', G_2)$ or $e(A'D', X) \neq e(C', G_2)$ **return** 0 **else return** 1

$\mathsf{CDLLink}((bsn_0, m_0, \Omega_0), (bsn_1, m_1, \Omega_1), gpk)$

For $b \in \{0, 1\}$ parse $\Omega_b = ({A'}_b, {B'}_b, {C'}_b, {D'}_b, J_b, \pi_b)$
if $bsn_0 \neq bsn_1$ or $bsn_0, bsn_1 = \perp$ **return** 0
if $\exists b \in \{0, 1\}$ such that $\mathsf{CDLVerify}(bsn_b, m_b, \Omega_b, gpk) = 0$ **return** 0
if $J_0 = J_1$ **return** 1 **else return** 0

$\mathsf{CDLIdentify}_T(\mathcal{T}, (f, A, B, C, D))$

Let $\mathcal{T} = (n, (F, \pi_f), (cre, \pi_{cre}))$, **if** $F = G_1^f$ and $cre = (A, B, C, D)$ **return** 1 **else return** 0

$\mathsf{CDLIdentify}_S(bsn, m, \Omega, (f, A, B, C, D))$

Let $\Omega = (A', B', C', D', J, \pi)$, **if** $\mathsf{CDLVerify}(bsn, m, \Omega, gpk) = 0$ **return** 0
if $D' = {B'}^f$ **return** 1 **else return** 0

Figure 2.8: The algorithms of CDL

$\mathsf{CDLJoin}(gpk)$	$\rightleftharpoons$	$\mathsf{CDLIssue}(isk, gpk, uid)$
		choose $n \leftarrow_{\$} \{0,1\}^{\tau}$
	$\xleftarrow{n}$	
$f \leftarrow_{\$} \mathbb{Z}_p^*, F \leftarrow G_1^f,$		
$\pi_F \leftarrow_{\$} \mathsf{SPK}\{(\underline{f}) : F = G_1^f\}(n)$		
	$\xrightarrow{F, \pi_F}$	
		Verify π_F, $F \in \mathbb{G}_1$ and $F \neq 1$
		$s \leftarrow_{\$} \mathbb{Z}_p^*, A \leftarrow {G_1}^s, B \leftarrow A^{\beta}$
		$C \leftarrow A^{\alpha} F^{s\alpha\beta}, D \leftarrow F^{s\beta}$
		$cre \leftarrow (A, B, C, D), \pi_{cre} \leftarrow_{\$}$
		$\mathsf{SPK}\{t : B = G_1^t \wedge D = F^t\}$
		$\mathbf{reg}[uid] \leftarrow (F, cre)$
	$\xleftarrow{cre, \pi_{cre}}$	
if $A = 1$ **return** $\perp$		
Verify π_{cre}		
if $e(A, Y) \neq e(B, G_2)$ or $e(AD, X) \neq e(C, G_2)$		
return $\perp$		
$\mathbf{gsk}[uid] \leftarrow (f, cre)$		

Figure 2.9: The $<$ CDLJoin, CDLIssue $>$ Protocol

Chapter 3

Modelling Centralised Reputation Systems with Unlinkable User Behaviour

Contents

3.1 Introduction

This chapter introduces a new model for centralised reputation systems and a construction satisfying this model using two variants of a group signature scheme as building blocks.

3.1.1 Motivation

We first focus on an application of group signature schemes to *centralised reputation systems.*

Reputation has always played a fundamental role in how we exchange products and services. While traditionally we have been used to trusting the reputation of established brands or companies, we are now facing a new challenge in the online world: determining the trustworthiness of a wide variety of possible exchanges. Whether we are selecting a restaurant, buying a product or getting a taxi, we are increasingly relying on scores and ratings to make our choice. For example, on Amazon, which in 2015 had over 2 million third party sellers worldwide [1], each seller is given a rating out of 5. Also Uber, with over 40 million monthly active users [3], allows drivers and passengers to rate each other.

A *reputation system* formalises this process of rating a user or service by associating them with a *reputation value* representing their trustworthiness. A reputation is then built as the value gets updated over time, as a consequence of user interactions and service exchanges.

There are several privacy implications of reputation systems. Obviously, to form a reputation value for a specific user or service, their behaviour across interactions needs to be linked. However, a user could be deanonymised by linking all their interactions together in a profiling attack. Also the detection of multiple feedback on the same instance also involves linking users' feedback. Given this, a cryptographic treatment of reputation systems has been considered necessary, and several models have been proposed in the literature so far [7, 18, 58].

Reputation systems can be generally categorised into *distributed* or *centralised* systems. Distributed systems [101] have no central server and use local reputation values, i.e., reputation values created by users on other users. For example, a user may generate a reputation value based on feedback from querying other users, and their own interactions. This means a user does not have a unique reputation value, but many other users hold their own reputation value for them. As users must form reputation values on others, their interactions cannot be anonymous. Examples of distributed reputation systems are *decentralised reputation systems* (DRS) [51], based on querying other nodes, and *privacy*

preserving DRS [78, 111], designed to maintain anonymity when answering queries from other nodes using multi-party computation.

Centralised reputation systems, on the other hand, have a *central server* that manages the network, performing tasks such as controlling communication between users, receiving feedback and evaluating reputation values. We will focus on centralised systems since the reputation systems used by most service providers such as Airbnb, Uber and Amazon are of this type. A variety of centralised reputation system models have been proposed in the literature. While their applications and the used techniques vary greatly, all of the models have in common that reputations are assigned to each *item* or service, the object of the reputation, rather than each *user*, the provider of the service. To understand the limitations of this, let us consider the case of online shopping: in such a scenario, existing reputation systems would typically allocate a reputation to each product sold (item), and not to each seller (user), based on all their sold products.

In this chapter, we advocate the need for a shift in how reputation systems are modelled, and we propose a model for reputation systems in which a reputation value is given to each user, based on all their user behaviour or items. This is crucial in ensuring that a user's previous behaviour will contribute to their current reputation, instead of having separate reputations for each service provided. Clearly, if items belonging to a user could be linked together, the model which has been used so far could be transformed into our new one, by collating the reviews for each item belonging to a user to form a reputation value. However, if the user wishes to make their items unlinkable for privacy reasons, then this becomes more challenging. We note that users can refer to both consumers and providers of a service. However, as many applications such as Uber and AirBnb allow both consumers and providers to be rated, we do not distinguish between the two.

A car pooling app is an example of the reputation systems we are modelling. A user may not want their trips (or items) to be linked together, as their movements could be tracked. However, a user's reputation should be based on their previous trips, so others can judge their reliability. In this context, reputations based on each journey are not useful, as they cannot be used for future journeys. This is why it is important to assign reputation values to users instead of items.

3.1.2 Existing Work

There exist several security models for centralised reputation systems. Reputation systems using ecash for reputation points were modelled in [7], where a reputation value corresponds to the number of a user's reputation points. However, this model is limited since it only captures reputation systems where reputations are formed in this way.

The model proposed in [18] is inspired by the security model for dynamic group signatures [14], and the authors provide an extra linkability feature to detect users giving multiple feedback on the same subject. In [58], the security requirements for this model are improved by giving more power to the adversary, (for example, in the public linkability security requirement the key issuer is no longer assumed to be honest), and introducing the requirement that an adversary cannot forge feedback that will link to another user, invalidating their feedback. In [58] the model is also made fully dynamic [24], i.e. users can join or leave the scheme at any time, and a lattice-based instantiation satisfying this model is provided.

Crucially, in both [18] and [58] reputations are assigned to each *item*, the subject of feedback, not each *user*. In contrast, we propose a new model for reputation systems in which a reputation value is given to each user, based on all their user behaviour or items. This ensures a user's reputation reflects their entire past behaviour and so ensures they are accountable for their previous actions, modelling more accurately how such systems truly operate.

3.1.3 Our Contribution

Our contribution is to propose a new model for reputation systems so that reputation values are given to users instead of items, whilst guaranteeing that the user's behaviour is unlinkable, and that the central server does not have to be involved during every transaction. This means that users can have multiple unlinkable items, whilst a reputation value still reflects their entire behaviour. Therefore users can have the benefits of privacy, whilst still being held accountable.

The first challenge when developing such reputation systems, is to provide a mechanism for

generating reputation values, whilst ensuring items cannot be linked by user. We model this with a ReceiveFB algorithm run by the central server ($\mathcal{CS}$), which takes feedback, and links it to other feedback on items with the same author, updating their reputation. We define the security requirement, *unlinkability of user behaviour*, which defines the unlinkability of items by the same user achievable while reputations can still be updated using ReceiveFB. Our approach as described so far gives rise to a possible attack in which a user produces a valid item which will not contribute to this user's reputation, or will even unfairly affect another user in ReceiveFB. We introduce the *traceability* security requirement to mitigate against this attack. These security requirements are reminiscent of those for group signature schemes in [12].

The second challenge is to determine the reputation of a specific user, whose items are unlinkable. A naive solution could be for the user to simply attach their reputation to an item, but the user could lie about their reputation. To avoid this, we introduce the PostItem algorithm, with which the user posts their item, and proves they were given a reputation at a particular time, using a token generated by the $\mathcal{CS}$. We further introduce the security requirement of *unforgeability of reputation* to ensure the user cannot lie about their reputation.

Finally, the standard security requirements of a centralised reputation system [18, 58], namely *anonymity of feedback*, *soundness of reputation* and *non–frameability*, still need to hold, and we adapt these naturally to our new model.

We then show how such a reputation system satisfying our security requirements is achievable using two variants of group signature schemes: a group signature scheme that has been bound to reputation for sending items, and a DAA scheme [29] for sending feedback. We present a concrete construction which ensures the unlinkability of user behaviour, traceability and unforgeability of reputation requirements. Our modification to [53], similarly to in [106], allows users to prove their reputation at a particular time. Our construction also makes use of the DAA scheme given in [33], which ensures anonymity of feedback, whilst multiple feedback on the same item can be detected, ensuring soundness of reputation. This is due to the *user controlled linkability* property of the DAA scheme, where only signatures with the same *basenames* can be linked. We set the basename to be the item feedback is given on.

To reduce complexity, our model is in the *static* setting, however it could simply be converted to the *dynamic* case using [14]. Our construction uses building blocks designed in the dynamic case and so could also be easily converted.

Although our scheme aims to provide reputation values for users, it would be straightforward to provide reputation values for items as well. The Central Server could simply keep track of all feedback given on an item and publish the item alongside its reputation based on just this feedback. In many applications, such as AirBnb, reputations are provided for both users and items.

3.2 Chapter Preliminaries

3.2.1 Existing Work on Centralised Reputation Systems

We now provide an overview of several centralised reputation systems that have been proposed for different applications that are not proven secure under a particular cryptographic model for a reputation system.

The reputation system *PerChatRep* [126] was developed for pervasive social chatting; users chat with each other and provide feedback on this. This scheme is a hybrid between a distributed and centralised system. Local reputations are used, as well as a global reputation calculated by the central server. The scheme manages privacy by changing the pseudonyms used by nodes, at regular intervals. Therefore it does not achieve unlinkability of all of a user's behaviour to other users. This is also true for sending feedback.

AdContRep [125] is a reputation system that works in a similar way to PerChatRep but is specifically for MANET content services, which help nodes to decide which service provider to use for different applications. In this scheme reputations for different content as well as users are tracked. *Mobile ad hoc networks* (MANETs) are networks of mobile devices that communicate wirelessly, in a decentralised manner. However, nodes do communicate with a central server periodically in AdContRep.

The reputation system *AnonRep* was devised in [128] for use in internet message boards with anonymity. In order to form reputation values, messages must be opened to reveal

the author. Anonrep uses *multi–party computation* to restrict this opening power, so that reputation values are formed by a group of servers, instead of one trusted party. Users' messages can only be de-anonymised if all servers collude.

In the context of *vehicular ad-hoc networks* (VANETs), networks of vehicles that allow communication, an example is the reputation system in [106], where vehicles communicate directly, and messages and feedback are anonymous and unlinkable. This reputation system uses a modification of the BBS group signature scheme [21], which we will call *BBS**, that allows users to prove their reputation. Finally, [70] fixes a bug in the BBS* scheme, and uses BBS* and a modified direct anonymous attestation (DAA) scheme to build a reputation system for pervasive social networks with stronger anonymity for feedback than [106].

In [112], a reputation system is proposed that ensures feedback remains anonymous, while ensuring feedback can only be given after the entity that is being rated confirms a transaction happens with the rater. This ensures reliability of reputation values by ensuring that users cannot collude to repeatedly unfairly rate another user.

In [16], *signatures of reputation* are proposed, which allow users to sign demonstrating they have a particular reputation, provided the measure of reputation is *monotonic*. This means that extra feedback only increases the reputation score. A cryptographic security model is given for this primitive, but not the wider reputation system.

3.2.2 Conventional attacks on Reputation Systems

We now detail several attacks on reputation systems outside the scope of a cryptographic model. These attacks can be mitigated in other ways when a reputation system is implemented by making careful design choices. We will evaluate how susceptible our construction is to these attacks at the end of this chapter.

- **The On–Off attack**: In this attack [95] adversaries behave honestly for some time, building up a good reputation. They can then start behaving dishonestly. Their high reputation enables them to do this effectively and for a long period of time, before their reputation drops. This is particularly effective for an adversary with

one specific target in mind. For example in eBay, a user could build up a good reputation score for small orders in order to later defraud users during a big order. This is also referred to as the *traitors* attack [102]. This attack can be mitigated by adjusting the weighting of the final reputation formation, so that bad behaviour will cause the reputation to deteriorate quickly. This ensures that more recent feedback will have more value.

- **Unfair Rating attack**: A malicious node repeatedly provides dishonest feedback (without collaboration) [54]. This unfairly affects the other user's reputation. This attack can be partly mitigated by ensuring each user can only provide feedback once for each instance.

- **The Collaborative Unfair Rating attack**: This attack, described in [95] as the *collaborative bad mouthing attack*, involves many malicious users colluding together in the unfair rating attack. They all collude to submit dishonest feedback on a user. This is also referred to as a *colluders attack* [102]. A form of this attack is the *front peers attack* [102], when malicious users collude to increase each other's reputations. For example, they might post a public message, and inform all other users to submit positive feedback on this message. In *sybil attacks* these colluding users are fake identities.

  This attack can be mitigated by ensuring only one feedback can be given per instance, and also by making joining the scheme expensive in some way. New users to the scheme could be verified. Another way is through incentivising honest feedback, which was shown game theoretically to ensure players rate accurately [123].

- **The Whitewashers attack**: *Whitewashers* are malicious users that leave and then rejoin the reputation scheme in order to lose their bad reputation [102]. After a user has behaved maliciously they leave the system and then rejoin under a new identity with a new reputation score. A way to mitigate this attack is to assign new users a low reputation so that they must prove themselves in the network, as well as making joining the scheme expensive in some way.

- **The Self–Rating attack**: Adversaries positively rate a large number of their own items. This is particularly effective if they can create a large number of fake items to positively rate. Ideally this attack should be avoided by preventing the user from self rating. The central server can also punish authors of items that do not represent a valid transaction.

3.3 Defining a Reputation System

We first introduce the syntax and generic functionality of a centralised reputation system RS with unlinkable user behaviour, and a reputation value assigned to users.

We define a reputation system, RS, as consisting of the following probabilistic polynomial–time algorithms: Setup, AllocateReputation, PostItem, CheckItem, SendFB, VerifyFB, LinkFB, ReceiveFB. We illustrate our model in Figure 3.1.

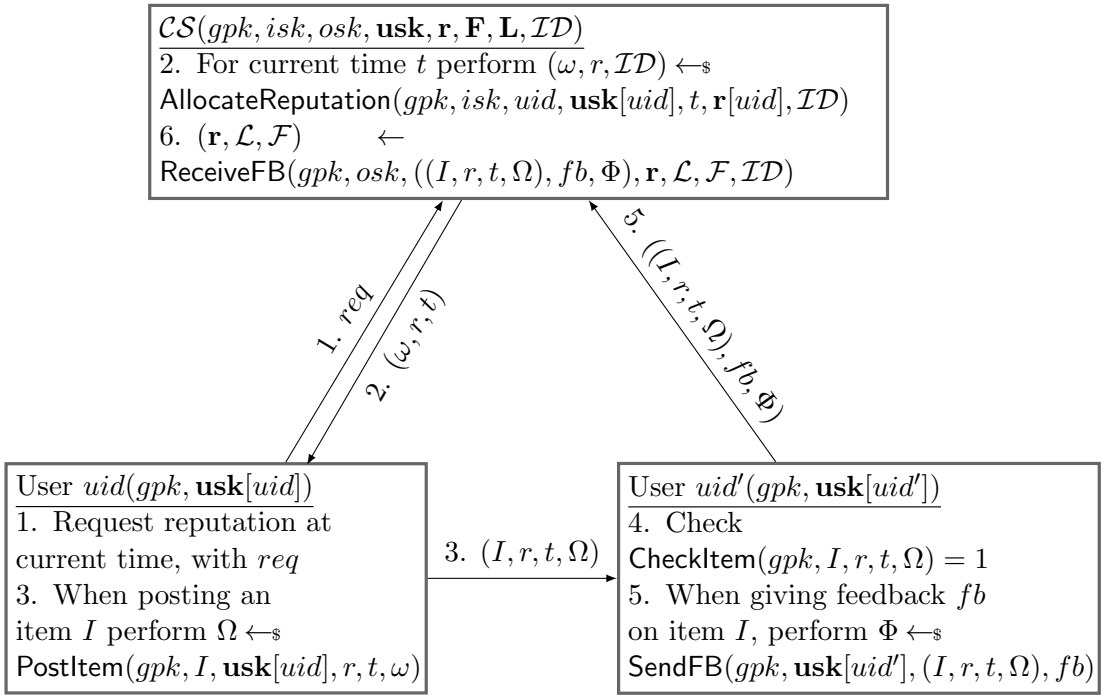

Figure 3.1: How entities interact in our model of a centralised reputation system

SendFB, VerifyFB, and LinkFB are equivalent to the Sign, Verify, Link algorithms in [18] and [58]. The additional algorithms, represent the key features of our new approach.

We will refer to our model for a centralised reputation system as the RS model. The entities involved are a set of users $\mathcal{U} = \{uid_i\}$ and a *central server* ($\mathcal{CS}$). The $\mathcal{CS}$ has two secret keys, isk and osk. The *issuing secret key*, isk, is necessary for allowing users to join the system and allocating them tokens to prove their reputation, whereas the *opening secret key*, osk, is necessary for forming reputations from feedback. For simplicity we provide the $\mathcal{CS}$ with both secret keys, but to reduce the power of one entity the role of the $\mathcal{CS}$ could be distributed.

The $\mathcal{CS}$ begins by running Setup. Users post items[1], which are the subject of feedback, while proving their reputation at a certain time using PostItem. After a request from a user, the $\mathcal{CS}$ runs AllocateReputation, which outputs tokens to allow a user to prove their current reputation at a specific time in PostItem. Other users verify that an item is a valid output of PostItem by running CheckItem. This ensures the item was authored by an enrolled user, the reputation alongside the item is correct for the given time, and the $\mathcal{CS}$ can use feedback on the item to form reputations. SendFB is run by a user when giving feedback on an item, and its output is sent to the $\mathcal{CS}$. ReceiveFB is run by the $\mathcal{CS}$ when receiving the output of SendFB from a user. The $\mathcal{CS}$ updates their stored feedback and reputations, based on this. VerifyFB and LinkFB are used by ReceiveFB to check the feedback is valid and that there is no feedback by the same user on this item, otherwise ReceiveFB will abort.

We note that in our model, any user can give feedback on an item. This is useful for applications where items are publicly posted and rated, such as a message boards. However, for applications where only a user who has consumed a service corresponding to a particular item should be able to rate this item, care must be taken when sharing items. Only the users who are allowed to rate an item should be given the item.

In the car pooling example, whenever a driver wishes to update their reputation, they request the $\mathcal{CS}$ run AllocateReputation to obtain a token for their reputation. They are incentivised to do this by the fact the reputation is displayed alongside the time it was allocated. When they wish to offer a ride, they use their most recent token to share an item with these passengers through PostItem, which can be verified by passengers with CheckItem. The passenger can then pay using some anonymous payment system. After the ride, their passenger can then provide feedback on this item to the $\mathcal{CS}$ using SendFB. The $\mathcal{CS}$ uses ReceiveFB to update their lists of feedback and reputations for each user, if the feedback is valid.

Before describing in detail our new model, we provide an overview of our notation in Table 3.1.

We keep $\mathcal{R}$, Aggr, $\hat{r}$ generic to ensure a flexible model that applies to different applications. We now state some examples.

[1] A simple example of an item could be a product being sold.

$\mathcal{R}$	The set of all possible reputation values.
$\hat{r}$	The initial reputation of every user at the system's setup.
$\mathcal{U}$	The set of all users in the scheme.
Aggr	A function that takes as input the new feedback fb, the user whose reputation is being updated uid, the list of feedback already received $\mathcal{F}$, and the most recent reputation r, and outputs the new reputation r'.
$\mathbf{r}$	For the user $uid \in \mathcal{U}$, $\mathbf{r}[uid]$ is the user uid's reputation held by the $\mathcal{CS}$.
$\mathcal{L}$	A list of feedback that will contain entries in the form of a 6-tuple $((I, r, t, \Omega), fb, \Phi)$, where (fb, Φ) is feedback/ proof pair, given on item I with reputation r and time t, with the proof Ω. $\mathcal{L}$ is used by the $\mathcal{CS}$ to keep track of all feedback given, so that multiple feedback on the same item can be detected in ReceiveFB.
$\mathcal{F}$	A list of feedback that will contain entries of the form (uid, fb) where fb is feedback given to user uid. $\mathcal{F}$ is used by the $\mathcal{CS}$ to keep track of all feedback given on user uid to form reputations in ReceiveFB.
$\mathcal{ID}$	A list of identities for all users, this list will allow the $\mathcal{CS}$ to store information on users whilst running AllocateReputation for use in ReceiveFB.

Table 3.1: An overview of the notation used in our RS model

In an application that values anonymity over a fine grained reputation, the set $\mathcal{R}$ could consist of reputation values $\{0, 1, 2, 3, 4, 5\}$. In an application where fine grained reputation is valued over large anonymity sets, the set $\mathcal{R}$ could consist of reputation values $\{n/10 : n \in [0, 500]\}$.

The initial reputation $\hat{r}$ could be set to 0 or to an average value. The choice of 0 could potentially penalise new users, but the choice of an average value could allow users to increase their reputation by rejoining the scheme.

If feedback is in the same set as $\mathcal{R}$, then an example of Aggr that would fit with the previous examples is:

$$\mathsf{Aggr}(fb, uid, \mathcal{F}, r) = \frac{fb + kr}{1 + k}, \text{ such that } k = |\{(uid, fb') \in \mathcal{F}\})|.$$

For this example, reputation values are the average of all feedback received. Alternatively, this average could be weighted so that more recent feedback counts for more than more distant feedback.

3.3.1 Syntax of RS

We now present the syntax for our RS model of a centralised reputation system. We note that VerifyFB and LinkFB are never used explicitly, but only used internally by ReceiveFB. However, as in existing work [18, 58], we include these algorithms separately to allow for notational simplicity when defining security.

Definition 3.1 (RS)**.** A centralised reputation system RS consists of the following algorithms:

Setup$(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) \to (gpk, isk, osk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$**:** takes as input security parameter τ, a set $\mathcal{R}$ of reputation values, $\hat{r} \in \mathcal{R}$, the initial reputation, a set of users $\mathcal{U}$, and the aggregation algorithm Aggr. The $\mathcal{CS}$ computes a public key gpk, the issuing secret key isk, which is used to issue new user secret keys and in AllocateReputation, and the opening secret key osk, which is used in ReceiveFB to trace the author of an item to form reputations. The $\mathcal{CS}$ computes a secret key for each user, $\mathbf{usk} = \{\mathbf{usk}[uid] : uid \in \mathcal{U}\}$, and $\mathbf{r}$, the reputation for all users held by the $\mathcal{CS}$, where $\forall uid \in \mathcal{U}, \mathbf{r}[uid] = \hat{r}$. The $\mathcal{CS}$ creates empty lists $\mathcal{L}, \mathcal{F}, \mathcal{ID}$.

AllocateReputation$(gpk, isk, uid, \mathbf{usk}[uid], t, \mathbf{r}[uid], \mathcal{ID}) \to (\omega, \mathbf{r}[uid], \mathcal{ID})$**:** takes as input the public key gpk, the issuing secret key isk, user uid's secret key $\mathbf{usk}[uid]$, the current time t, the current reputation of user uid held by the $\mathcal{CS}$ $\mathbf{r}[uid]$, and the list of identities for users $\mathcal{ID}$. It updates the list of identities $\mathcal{ID}$, and generates ω which allows user uid to prove they have reputation $\mathbf{r}[uid]$.

PostItem$(gpk, I, \mathbf{usk}[uid], r, t, \omega) \to \Omega$**:** takes as input the public key gpk, an item I, user uid's secret key $\mathbf{usk}[uid]$, the last reputation r, time t and token ω received from the $\mathcal{CS}$ (r is not necessarily the reputation $\mathbf{r}[uid]$ held by the $\mathcal{CS}$). It outputs Ω, which proves the author is enrolled and has reputation r at time t, and is used in ReceiveFB to form a reputation for user uid.

CheckItem$(gpk, I, r, t, \Omega) \to \{0, 1\}$**:** takes as input the public key gpk, an item I, a reputation r, a time t and Ω. It outputs 1 if Ω is a valid output of PostItem, given (I, r, t), and 0 otherwise.

SendFB$(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb) \to \Phi$**:** takes as input the public key gpk, user uid's secret key $\mathbf{usk}[uid]$, the subject of their feedback, (I, r, t, Ω), and the feedback fb.

It outputs Φ which is sent to the $\mathcal{CS}$, to prove the author of Φ is enrolled, and also for the detection of multiple feedback.

VerifyFB$(gpk, (I, r, t, \Omega), fb, \Phi) \rightarrow \{0, 1\}$**:** takes as input the public key gpk, an item (I, r, t, Ω), and feedback/ proof pair on this item (fb, Φ). It outputs 1 if Φ is a valid output of SendFB, and 0 otherwise.

LinkFB$(gpk, (I, r, t, \Omega), fb_0, \Phi_0, fb_1, \Phi_1) \rightarrow \{0, 1\}$**:** takes as input the public key gpk, an item (I, r, t, Ω), and two feedback/ proof pairs on this item, (fb_0, Φ_0), (fb_1, Φ_1). It outputs 1 if Φ_0 and Φ_1 were generated by the same user with the same input of (I, r, t, Ω), and 0 otherwise.

ReceiveFB$(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID}) \rightarrow (\mathbf{r}, \mathcal{L}, \mathcal{F})$**:** takes as input the public key gpk, the opening secret key osk, a feedback/ proof pair (fb, Φ) on item (I, r, t, Ω), the current reputations $\mathbf{r}$ held by the $\mathcal{CS}$, the lists of feedback so far $\mathcal{L}$ and $\mathcal{F}$, and the list of user identities $\mathcal{ID}$. If Φ is not valid, or the LinkFB algorithm finds multiple feedbacks in $\mathcal{L}$ then it outputs $\perp$. Otherwise, it uses the aggregation algorithm Aggr, and the list $\mathcal{F}$, to update $\mathbf{r}$, $\mathcal{L}$ and $\mathcal{F}$ to take into account the new feedback.

3.4 Security Properties

We now present the desirable security and privacy properties for our RS model.

As discussed earlier, we consider reputation systems satisfying the following properties: correctness, unforgeability of reputation, traceability, unlinkability of user behaviour, soundness of reputation, anonymity of feedback, and non–frameability.

We introduce unforgeability of reputation, a requirement that ensures a user cannot prove that they have a reputation for a certain time, which differs from the one they were allocated by the $\mathcal{CS}$ in AllocateReputation. This is necessary because when an item is unlinkable, the author's reputation cannot be determined. Therefore the reputation must be included alongside the item. This property ensures that the sender has not lied about their reputation.

We also introduce unlinkability of user behaviour, which formalises our definition of unlinkable user behaviour, given that ReceiveFB can still form reputations, as well as traceability,

which ensures that all items generated by an adversary can be traced back to them when computing their reputation. This is necessary because, due to the unlinkability of user behaviour property, an attacker could attempt to subvert ReceiveFB. These properties are reminiscent of the full-anonymity and full-traceability properties [12] for static group signature schemes, and have been adapted for reputation systems. Traceability and Unforgeability of Reputation are separate requirements because they seek to ensure different properties: that user cannot lie about their reputation and that a user cannot avoid their behaviour being traced back to them in ReceiveFB.

Soundness of reputation ensures an adversary cannot submit multiple feedback on the same item, undermining the integrity of reputation values. Anonymity of feedback ensures that feedback cannot be traced to the user's identity and is unlinkable. We have adapted these two properties from [18] to fit our notation[2]. Non–frameability ensures that an adversary cannot produce a feedback that links to another user's feedback under LinkFB, and so unfairly stop this feedback being counted. We have adapted this property from [58]. Non–Frameability and Soundness of Reputation are separate requirements, because non–frameability captures that feedback should not be unfairly linked together, whereas Soundness of Reputation captures that feedback should not be unfairly unlinkable.

We highlight that the issuing secret key is used by the $\mathcal{CS}$ for joining users to the scheme, and therefore for the traceability and soundness of reputation property the adversary cannot corrupt the isk as otherwise they could cheat by creating unregistered users. The opening secret key is used by the $\mathcal{CS}$ to trace items, so that reputations can be updated with new feedback. Therefore in the unlinkability of user behaviour property the adversary cannot corrupt the osk as otherwise they could trace signatures. This means the $\mathcal{CS}$ could be split into two separate entities with different secret keys.

We trust that the $\mathcal{CS}$ updates users' reputations correctly with ReceiveFB, meaning the $\mathcal{CS}$ could potentially unfairly tamper with a user's reputation. This assumption is necessary because to publicly scrutinise the $\mathcal{CS}$, the de-anonymisation of items input to the $\mathcal{CS}$ would have to be made public. We justify this honest–but–curious trust assumption by remarking that the manager of a reputation system will have a vested interest in ensuring the fairness of the system to avoid disgruntled paying customers from moving elsewhere.

[2]Soundness of reputation is comparable to public linkability and anonymity of feedback is comparable to anonymity.

Oracles and State. The security notions that we will formalise use a number of oracles which keep joint state, e.g., by keeping track of queries and the set of corrupted parties. We present the detailed description of all oracles in Figure 3.2 and an overview of them and their maintained records.

USK (corrupts users) Corrupts user uid, and outputs their secret key $\mathbf{usk}[uid]$. Corrupted users are stored in $\mathcal{C}$.

POST (create items) This oracle returns valid items of a user uid.

SENDFB (create feedback) This oracle returns valid feedbacks of a user uid, storing outputs in the sets $\mathcal{G}_{uid}$, for use in the non–frameability property.

RECEIVEFB (perform ReceiveFB) This oracle returns outputs of the ReceiveFB algorithm.

ALLREP(perform AllocateReputation) This oracle returns outputs of the AllocateReputation algorithm.

$\underline{\mathsf{USK}(uid)\text{:}}$

$\mathcal{C} \leftarrow \mathcal{C} \cup \{uid\}; \mathbf{return}\ \mathbf{usk}[uid]$

$\underline{\mathsf{POST}(I, uid, r, t, \omega)\text{:}}$

$\mathbf{return}\ \mathsf{PostItem}(gpk, I, \mathbf{usk}[uid], r, t, \omega)$

$\underline{\mathsf{SENDFB}(uid, fb, (I, r, t, \Omega))\text{:}}$

$\Phi \leftarrow_{\$} \mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb), \mathcal{G}_{uid} \leftarrow \mathcal{G}_{uid} \cup \{((I, r, t, \Omega), fb, \Phi)\}$ $\mathbf{return}\ \Phi$

$\underline{\mathsf{RECEIVEFB}((I, r, t, \Omega), fb, \Phi, \mathcal{ID})\text{:}}$

$\mathbf{return}\ (\mathbf{r}, \mathcal{L}, \mathcal{F}) \leftarrow \mathsf{ReceiveFB}(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$

$\underline{\mathsf{ALLREP}(uid, t, r)\text{:}}$

$\mathbf{return}\ \mathsf{AllocateReputation}(gpk, isk, uid, \mathbf{usk}[uid], t, r, \mathcal{ID})$

Figure 3.2: Oracles in our RS security model

3.4.1 Correctness

The reputation system RS should function correctly. We formulate correctness via five conditions.

1. Items computed honestly via AllocateReputation and PostItem will verify correctly under CheckItem.

2. Feedbacks computed honestly via SendFB will verify correctly under VerifyFB.
3. On input outputs of SendFB on the same item of (I, r, t, Ω), using the same user secret key, the LinkFB algorithm will output 1.
4. If an item and feedback were generated honestly via PostItem and SendFB, then ReceiveFB updates $\mathbf{r}, \mathcal{L}, \mathcal{F}$ correctly.
5. If ReceiveFB is input feedback that is not valid under VerifyFB, or links to other feedback in $\mathcal{L}$ under LinkFB, then it will output $\perp$.

Definition 3.2 (Correctness)**.** A reputation system RS satisfies correctness if conditions 1–5 are satisfied.

Condition 1 is satisfied if for all $\tau \in \mathbb{N}$, all polynomially bounded $\mathcal{R}$,$\mathcal{U}$, $\hat{r} \in \mathcal{R}$, all aggregation algorithms Aggr, $(gpk, isk, \mathbf{usk}, \mathcal{ID}) \leftarrow\!\!\$\ \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$, all $uid \in \mathcal{U}$, any time $t \in \{0,1\}^*$, any $\mathbf{r} \in \mathcal{R}^{|\mathcal{U}|}$, $(\omega, r, \mathcal{ID}) \leftarrow\!\!\$\ \mathsf{AllocateReputation}(gpk, isk, uid, \mathbf{usk}[uid], t, \mathbf{r}[uid], \mathcal{ID})$, any $I \in \{0,1\}^*$, then

$$\mathsf{CheckItem}(gpk, I, r, t, \mathsf{PostItem}(gpk, I, \mathbf{usk}[uid], r, t, \omega)) = 1.$$

Condition 2 is satisfied if for all $\tau \in \mathbb{N}$, all polynomially bounded $\mathcal{R}$,$\mathcal{U}$, all $\hat{r} \in \mathcal{R}$, all aggregation algorithms Aggr, $(gpk, \mathbf{usk}) \leftarrow\!\!\$\ \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$, all $uid \in \mathcal{U}$, all $I \in \{0,1\}^*$, all $r \in \mathcal{R}$, all $t \in \{0,1\}^*$, all $\Omega \in \{0,1\}^*$, all $fb \in \{0,1\}^*$, then

$$\mathsf{VerifyFB}(gpk, (I, r, t, \Omega), fb, \mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb)) = 1.$$

Condition 3 is satisfied if for all $\tau \in \mathbb{N}$, all polynomially bounded $\mathcal{R}$,$\mathcal{U}$, all $\hat{r} \in \mathcal{R}$, all aggregation algorithms Aggr, $(gpk, \mathbf{usk}) \leftarrow\!\!\$\ \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$, all $uid \in \mathcal{U}$, all $I \in \{0,1\}^*$, all $r \in \mathcal{R}$, all $t \in \{0,1\}^*$, all $\Omega \in \{0,1\}^*$, all $fb_0, fb_1 \in \{0,1\}^*$, for $b \in \{0,1\}$, $\Phi_b \leftarrow\!\!\$\ \mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb_b)$, then

$$\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb_0, \Phi_0, fb_1, \Phi_1) = 1.$$

Condition 4 is satisfied if for all $\tau \in \mathbb{N}$, all polynomially bounded $\mathcal{R}$,$\mathcal{U}$, all $\hat{r} \in \mathcal{R}$, all aggregation algorithms Aggr, $(gpk, isk, osk, \mathbf{usk}, \mathcal{ID}) \leftarrow_{\$} \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$, all $uid, uid' \in \mathcal{U}$, $\mathbf{r} \in \mathcal{R}^{|\mathcal{U}|}$, all $t \in \{0,1\}^*$, $(\omega, r, \mathcal{ID}) \leftarrow_{\$} \mathsf{AllocateReputation}(gpk, isk, uid, \mathbf{usk}[uid], t, \mathbf{r}[uid], \mathcal{ID})$, all $I \in \{0,1\}^*$, $\Omega \leftarrow_{\$} \mathsf{PostItem}(gpk, I, \mathbf{usk}[uid], r, t, \omega)$, all $fb \in \{0,1\}^*$, $\Phi \leftarrow_{\$} \mathsf{SendFB}(gpk, \mathbf{usk}[uid'], (I, r, t, \Omega), fb)$, all $\mathcal{F}$, all $\mathcal{L}$ such that:

$\forall((I, r, t, \Omega), fb', \Phi') \in \mathcal{L}$: $\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb, \Phi, fb', \Phi') = 0$,

$(\mathbf{r}^*, \mathcal{L}^*, \mathcal{F}^*) \leftarrow \mathsf{ReceiveFB}(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$, then

$$(\mathbf{r}^*, \mathcal{L}^*, \mathcal{F}^*) \neq \perp, \mathcal{L}^* = ((I, r, t, \Omega), fb, \Phi) \cup \mathcal{L}, \mathcal{F}^* = \mathcal{F} \cup (uid, fb),$$

$$\mathbf{r}^*[uid] = \mathsf{Aggr}(fb, uid, \mathcal{F}, \mathbf{r}[uid]), \text{ and } \forall \hat{uid} \neq uid \in \mathcal{U}, \mathbf{r}^*[\hat{uid}] = \mathbf{r}[\hat{uid}].$$

Condition 5 is satisfied if for all $\tau \in \mathbb{N}$, all polynomially bounded $\mathcal{R}$,$\mathcal{U}$, all $\hat{r} \in \mathcal{R}$, all aggregation algorithms Aggr, $(gpk, osk, \mathbf{usk}, \mathcal{ID}) \leftarrow_{\$} \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$, all $\mathbf{r}$, all $\mathcal{F}$, $(\mathcal{L}, (I, r, t, \Omega), fb, \Phi)$ with $\mathsf{VerifyFB}(gpk, (I, r, t, \Omega), fb, \Phi) = 0$ or $\exists((I, r, t, \Omega), fb', \Phi') \in \mathcal{L}$ such that $\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb, \Phi, fb', \Phi') = 1$ then

$$\perp \leftarrow \mathsf{ReceiveFB}(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID}).$$

We now present our new security properties, which are necessary as reputation values are assigned to users instead of their individual unlinkable items.

3.4.2 Unforgeability of Reputation

A user can only prove that they have reputation r at time t, if this was allocated to them by the $\mathcal{CS}$ in $\mathsf{AllocateReputation}$. In the context of car pooling, this security property ensures a driver cannot lie about their reputation when requesting a passenger.

In our security game in Figure 3.3, the adversary is given the opening secret key osk, the list of user identities $\mathcal{ID}$, the USK, POST, ALLREP oracles, but not isk, as they could run $\mathsf{AllocateReputation}$. The adversary wins if they output a valid item, for reputation r, time t, tracing to a corrupted user uid in $\mathsf{ReceiveFB}$, without querying (uid, r, t) to the ALLREP

Experiment: $\mathbf{Exp}^{anon-ub-b}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$

$(gpk, isk, osk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID}) \leftarrow_\$ \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$
$(\mathsf{st}, uid_0, uid_1, I, r, t, \mathcal{ID}) \leftarrow_\$ \mathcal{A}^{\mathsf{RECEIVEFB}}(\mathsf{choose}, gpk, isk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$
$\forall \tilde{b} \in \{0,1\} \quad (\omega_{\tilde{b}}, \mathcal{ID}) \leftarrow_\$ \mathsf{AllocateReputation}(gpk, isk, uid, \mathbf{usk}[uid_{\tilde{b}}], t, r, \mathcal{ID})$
$\Omega \leftarrow_\$ \mathsf{PostItem}(gpk, I, \mathbf{usk}[uid_b], r, t, \omega_b)$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{RECEIVEFB}}(\mathsf{guess}, \mathsf{st}, \Omega, \mathcal{ID})$
if $((I, r, t, \Omega), \cdot)$ queried to the RECEIVEFB oracle **return** 0
return b^*

Experiment: $\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$

$(gpk, isk, osk, \mathbf{usk}, \mathcal{ID}) \leftarrow_\$ \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}); C \leftarrow \emptyset$
$(I, r, t, \Omega, fb, \Phi, \mathbf{r}, \mathcal{L}, \mathcal{F}) \leftarrow_\$ \mathcal{A}^{\mathsf{USK,POST,SENDFB,ALLREP}}(gpk, osk, \mathcal{ID})$
if $\mathsf{CheckItem}(gpk, I, r, t, \Omega) = 0$ or $\mathsf{VerifyFB}(gpk, (I, r, t, \Omega), fb, \Phi) = 0$ **return** 0
if $\exists((I, r, t, \Omega), fb', \Phi') \in \mathcal{L}$ with $\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb, \Phi, fb', \Phi') = 1$ **return** 0
$\forall uid \in \mathcal{U}, \mathcal{ID} \leftarrow_\$ \mathsf{AllocateReputation}(gpk, isk, uid, \mathbf{usk}[uid], t, r, \mathcal{ID})$
if $\perp \leftarrow \mathsf{ReceiveFB}(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$ **return** 1
else $(\mathbf{r}^*, \mathcal{L}^*, \mathcal{F}^*) \leftarrow \mathsf{ReceiveFB}(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$
if $\mathcal{L}^* \neq ((I, r, t, \Omega), fb, \Phi) \cup \mathcal{L}$ **return** 1
if $\mathcal{F}^* = (uid', fb) \cup \mathcal{F}$ for some $uid' \in \mathcal{U} \quad uid^* \leftarrow uid'$ **else return** 1
if $\mathbf{r}^*[uid^*] \neq \mathsf{Aggr}(fb, uid^*, \mathcal{F}, \mathbf{r}[uid^*])$, or $\exists \hat{uid} \in \mathcal{U} \backslash \{uid^*\}$ such that $\mathbf{r}^*[\hat{uid}] \neq \mathbf{r}[\hat{uid}]$ **return** 1
if $uid^* \notin \mathcal{C}$ and $(I, uid^*, r, t, \cdot)$ was not queried to the POST oracle **return** 1
else **return** 0

Experiment: $\mathbf{Exp}^{unforge-rep}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$

$(gpk, isk, osk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID}) \leftarrow_\$ \mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$
$(I, r, t, \Omega) \leftarrow_\$ \mathcal{A}^{\mathsf{USK,POST,ALLREP}}(gpk, osk, \mathcal{ID})$
if Ω returned by POST or **if** $\mathsf{CheckItem}(gpk, I, r, t, \Omega) = 0$ **return** 0
$uid' \leftarrow_\$ \mathcal{U}, fb \leftarrow 0, \Phi \leftarrow_\$ \mathsf{SendFB}(gpk, \mathbf{usk}[uid'], (I, r, t, \Omega), fb)$
$(\mathbf{r}^*, \mathcal{L}^*, \mathcal{F}^*) \leftarrow \mathsf{ReceiveFB}(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$
if $\mathcal{F}^* \backslash \mathcal{F} = \{uid, fb\}$ for some $uid \in \mathcal{U} \quad uid^* \leftarrow uid$ **else return** 1
if $\mathcal{A}$ queried (uid^*, t, r) to ALLREP oracle and $uid^* \in \mathcal{C}$ **return** 0 **else return** 1

Figure 3.3: Experiments capturing our unlinkability of user behaviour, traceability and unforgeability of reputation security properties

oracle, or it does not trace to any user.

Definition 3.3 (Unforgeability of Reputation)**.** A reputation system RS satisfies unforgeability of reputation if for all polynomial–time adversaries $\mathcal{A}$, all sets $\mathcal{R}$ and $\mathcal{U}$ such that $|\mathcal{R}|$ and $|\mathcal{U}|$ are polynomially bounded in τ, all $\hat{r} \in \mathcal{R}$, all Aggr functions, the advantage $\Pr[\mathbf{Exp}^{unforge-rep}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1]$ is negligible in τ.

3.4.3 Traceability of users

This security property ensures that any valid item an adversary produces will contribute towards their own reputation in ReceiveFB. This also guarantees unforgeability. In the context of car pooling, this security property means that feedback on a driver's rides will always affect their own reputation and not another's.

In our security game in Figure 3.3, the adversary is given the opening secret key osk, the list of user identities $\mathcal{ID}$, the USK oracle to corrupt users, and the POST, SENDFB, ALLREP oracles for uncorrupted user, but not isk, because they could cheat by generating the secret key of a new user. They must output a valid item and feedback, and $\mathbf{r}, \mathcal{L}, \mathcal{F}$, such that the feedback does not link to any in $\mathcal{L}$. If ReceiveFB fails, does not correctly update $\mathbf{r}, \mathcal{L}, \mathcal{F}$, or updates the reputation of an uncorrupted user, then the adversary wins.

Definition 3.4 (Traceability)**.** A reputation system RS satisfies traceability if for all polynomial–time adversaries $\mathcal{A}$, all sets $\mathcal{R}$ and $\mathcal{U}$ such that $|\mathcal{R}|$ and $|\mathcal{U}|$ are polynomially bounded in τ, all $\hat{r} \in \mathcal{R}$, all Aggr functions, the advantage $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1]$ is negligible in τ.

3.4.4 Unlinkability of User Behaviour

This property ensures other users cannot link together items by author, while the $\mathcal{CS}$ can still link items to form reputation values based on a user's entire behaviour. In the context of car pooling, this security property means that all rides a driver/ user undertakes are unlinkable, so their movements cannot be tracked.

In our security game in Figure 3.3, the adversary is given all user secret keys, the issuing secret key isk, $\mathbf{r}$, $\mathcal{L}$, $\mathcal{F}$, and $\mathcal{ID}$, but not the opening secret key osk, because otherwise they could run ReceiveFB, and then check which user's reputation changes. They are given the RECEIVEFB oracle, but its use is restricted so that the challenge signature cannot be queried, to avoid this attack. This attack would not be practical in the real world, as reputations will be updated at intervals so that multiple users' reputations will change at once. Future work could consider specific Aggr algorithms that would allow this security property to be strengthened. In our work, to ensure our model is generic, we define security for all possible Aggr functions.

The adversary chooses an item I, a reputation r and a time t, an updated list of identities $\mathcal{ID}$, and two users uid_0, uid_1, they then must decide whether an item Ω was authored by uid_0 or uid_1.

Definition 3.5 (Unlinkability of User Behaviour)**.** A reputation system RS satisfies unlinkability of user behaviour if for all polynomial–time adversaries $\mathcal{A}$, all sets $\mathcal{R}$ and $\mathcal{U}$ such that $|\mathcal{R}|$ and $|\mathcal{U}|$ are polynomially bounded in τ, all $\hat{r} \in \mathcal{R}$, and all Aggr functions, the following advantage is negligible in τ :

$$\left|\Pr[\mathbf{Exp}^{anon-ub-0}_{\mathcal{A},\mathsf{RS}}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr})=1]-\Pr[\mathbf{Exp}^{anon-ub-1}_{\mathcal{A},\mathsf{RS}}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr})=1]\right|.$$

Experiment: $\mathbf{Exp}^{sound-rep}_{\mathcal{A},\mathsf{RS}}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr}))$

$(gpk, isk, osk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID}) \leftarrow_{\$} \mathsf{Setup}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr})$; $\mathcal{C} \leftarrow \emptyset$
$((I,r,t,\Omega),\{fb_j,\Phi_j\}_{j=1}^{|\mathcal{C}|+1}) \leftarrow_{\$} \mathcal{A}^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP}}(gpk,osk)$
if $\exists j \in [1,|\mathcal{C}|+1]$ such that $\mathsf{VerifyFB}(gpk,(I,r,t,\Omega),fb_j,\Phi_j)=0$ **return** 0
if $\exists j_1,j_2 \in [1,|\mathcal{C}|+1]$ with $j_1 \neq j_2$ s.t. $\mathsf{LinkFB}(gpk,(I,r,t,\Omega),fb_{j_1},\Phi_{j_1},fb_{j_2},\Phi_{j_2})=1$
return 0
if $\exists j \in [1,|\mathcal{C}|+1]$ s.t. $(uid, fb_j, (I,r,t,\omega))$ with $uid \notin \mathcal{C}$ was queried to the SENDFB oracle and Φ_j was returned **return** 0 **else return** 1

Experiment: $\mathbf{Exp}^{anon-fb-b}_{\mathcal{A},\mathsf{RS}}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr})$

$(gpk, isk, osk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID}) \leftarrow_{\$} \mathsf{Setup}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr})$
$(\mathsf{st}, uid_0, uid_1, fb, (I,r,t,\Omega)) \leftarrow_{\$} \mathcal{A}^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP}}(\mathsf{choose},gpk,isk,osk)$
$\Phi \leftarrow_{\$} \mathsf{SendFB}(gpk,\mathbf{usk}[uid_b],(I,r,t,\Omega),fb)$
$b^* \leftarrow_{\$} \mathcal{A}^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP}}(\mathsf{guess},\mathsf{st},\Phi)$
if uid_0 or uid_1 queried to the USK oracle **return** 0
if $(uid_0,\cdot,(I,r,t,\Omega))$ or $(uid_1,\cdot,(I,r,t,\Omega))$ queried to the SENDFB oracle **return** 0
return b^*

Experiment: $\mathbf{Exp}^{non-frame}_{\mathcal{A},\mathsf{RS}}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr}))$

$(gpk, isk, osk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID}) \leftarrow_{\$} \mathsf{Setup}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr})$; $\mathcal{C} \leftarrow \emptyset$; $\forall uid \in \mathcal{U}$ $\mathcal{G}_{uid} \leftarrow \emptyset$
$((I,r,t,\Omega),fb,\Phi) \leftarrow_{\$} \mathcal{A}^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP}}(gpk,isk,osk)$
if $\mathsf{VerifyFB}(gpk,(I,r,t,\Omega),fb,\Phi)=0$ **return** 0
if $\exists uid \in \mathcal{U}\backslash\mathcal{C}$ such that the following hold **return** 1 **else return** 0
$((I,r,t,\Omega),fb,\Phi) \notin \mathcal{G}_{uid}$ and
$\exists((I,r,t,\Omega),fb',\Phi') \in \mathcal{G}_{uid}$ with $\mathsf{LinkFB}(gpk,(I,r,t,\Omega),fb,\Phi,fb',\Phi')=1$

Figure 3.4: Experiments capturing our soundness of reputation, anonymity of feedback and non–frameability security properties

We now provide an overview of the existing security properties.

3.4.5 Soundness of Reputation Values

Users who are not enrolled should not be able to submit feedback. Reputation values should be based on only one piece of feedback per item per user. In the context of car pooling, this security property would mitigate against an attack where a passenger repeatedly gives feedback on one ride, unfairly negatively influencing the driver's reputation.

In the security game, adapted from [18], given in Figure 3.4, the adversary is able to corrupt users with the USK oracle, and is given the opening secret key osk, but not the issuing key isk, as they could use this to cheat by generating a secret key for a new user. They can use the SENDFB, ALLREP and POST oracles for uncorrupted users. The adversary outputs a list of feedback on the same item. They win if they can output more valid unlinkable feedback than the number of corrupted users, without using the SENDFB oracle.

Definition 3.6 (Soundness of Reputation)**.** A reputation system RS satisfies soundness of reputation if for all polynomial–time adversaries $\mathcal{A}$, all sets $\mathcal{R}$ and $\mathcal{U}$ such that $|\mathcal{R}|$ and $|\mathcal{U}|$ are polynomially bounded in τ, all $\hat{r} \in \mathcal{R}$, all Aggr functions, the advantage $\Pr[\mathbf{Exp}^{sound-rep}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1]$ is negligible in τ.

3.4.6 Anonymity of Feedback

Anonymity of feedback captures the anonymity of those providing feedback against the $\mathcal{CS}$, and up to all but two colluding users. Unfortunately it is not possible for a reputation system to have anonymity against all colluding users, whilst still satisfying soundness of reputation. This is because an adversary could discover whether a user uid authored some feedback $((I, r, t, \Omega), fb, \Phi)$ by running $\Phi' \leftarrow_{\$} \mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega),\ fb')$, then running $\mathsf{LinkFB}(gpk, (I, r,\ t, \Omega), fb, \Phi, fb', \Phi')$. If this outputs 1, then $((I, r, t, \Omega), fb, \Phi)$ must be authored by uid. In the context of car pooling, this security property means that provided passengers never submit multiple feedback on the same ride, their feedback will be unlinkable.

In the security game, adapted from [18], and given in Figure 3.4, the adversary is given isk, osk, and must choose two users uid_0 and uid_1, an item (I, r, t, Ω), and feedback fb. They then must decide which of these users authored the Φ returned to them. The adver-

sary can corrupt users with USK, and use SENDFB, POST and ALLREP for uncorrupted users. We do not allow the adversary to query uid_0 or uid_1 to the USK oracle, or to query SENDFB with either uid_0 or uid_1 and (I, r, t, Ω), so that they cannot perform the attack described.

Definition 3.7 (Anonymity of Feedback)**.** A reputation system RS satisfies anonymity of feedback if for all polynomial–time adversaries $\mathcal{A}$, all sets $\mathcal{R}$ and $\mathcal{U}$ such that $|\mathcal{R}|$ and $|\mathcal{U}|$ are polynomially bounded in τ, and all $\hat{r} \in \mathcal{R}$, all Aggr functions, the following advantage is negligible in τ :

$$\left|\Pr[\mathbf{Exp}^{anon-fb-0}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1] - \Pr[\mathbf{Exp}^{anon-fb-1}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1]\right|.$$

3.4.7 Non–frameability

This property, adapted from [58], ensures that an adversary, who has corrupted the central server and all users, cannot forge feedback that links to feedback of another user, meaning ReceiveFB detects multiple feedback by this user, and unfairly outputs $\perp$. In the context of car pooling, this security property means that a passenger cannot feedback on their own ride, linking to the driver involved, invalidating any feedback they submit.

In the security game, given in Figure 3.4, the adversary is given isk, osk and can corrupt users using the USK oracle, and use the POST, SENDFB, ALLREP oracles for uncorrupted users. To win, they must output valid feedback not output by the SENDFB oracle, which links to feedback output by the SENDFB oracle, authored by an uncorrupted user.

Definition 3.8 (Non–frameability)**.** A reputation system RS satisfies non–frameability if for all polynomial–time adversaries $\mathcal{A}$, all sets $\mathcal{R}$ and $\mathcal{U}$ such that $|\mathcal{R}|$ and $|\mathcal{U}|$ are polynomially bounded in τ, all $\hat{r} \in \mathcal{R}$, all Aggr functions, the advantage $\Pr[\mathbf{Exp}^{non-frame}_{\mathcal{A},\mathsf{RS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1]$ is negligible in τ.

3.5 A Centralised Reputation System with Unlinkable User Behaviour

We now present our construction that securely realises our RS model for reputation systems. Our construction makes use of two primitives: a modification of the XS group signature scheme XS*, and the CDL DAA scheme [33].

More specifically, we modify the XS group signature scheme [53], detailed in Section 2.7.2, similarly to the modification of the BBS group signature scheme [21] in [70, 106]. We use XS* for posting items in PostItem, CheckItem, and AllocateReputation. The XS scheme satisfies unlinkability of user behaviour, whilst still allowing reputations to be formed in ReceiveFB, using the opening key. Furthermore our modification allows a user to prove they were allocated a reputation at a certain time by AllocateReputation.

We then adopt the CDL [33] DAA scheme, detailed in Section 2.8.2, for the feedback component of the reputation system in SendFB, VerifyFB, LinkFB. This perfectly fits our requirements, because of the user controlled linkability of the DAA scheme. Signatures are signed with respect to a basename, and are linkable only when they have the same author and basename. Therefore in the context of reputation systems, by setting the basename to be the subject of the feedback, multiple feedback on the same item can be detected, whilst still ensuring anonymity of feedback.

3.5.1 Binding Reputation to the XS Group Signature Scheme

As discussed in the preliminaries, security requirements for group signatures were defined for static groups [12], dynamic groups [14], and fully dynamic groups [24]. The XS scheme [53] satisfies the security requirements for dynamic groups [14], of anonymity, traceability and non–frameability, under the q-SDH [19] assumption and in the random oracle model [13].

We present XS*, a modification of the XS scheme [53], to allow users to prove their reputation in PostItem. In this modification, we introduce an additional algorithm XSUpdate*, used in AllocateReputation, which outputs a token allowing a user to update their secret

$\mathsf{XSKeyGen}^*(gpp_1)$

$\xi_1, \xi_2 \leftarrow_\$ \mathbb{Z}_p; K \leftarrow_\$ \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}, H \leftarrow K^{\xi_1}, G \leftarrow K^{\xi_2}, \gamma \leftarrow_\$ \mathbb{Z}_p, W \leftarrow G_2^{\gamma}$
return $gpk_1 = (G_1, K, H, G, G_2, W), isk_1 = \gamma, osk = (\xi_1, \xi_2)$

$\mathsf{XSJoin}^*(isk_1,\ uid,\ gpk_1)$

$x, y \leftarrow_\$ \mathbb{Z}_p; \quad Z \leftarrow (G_1 H^y)^{\frac{1}{\gamma+x}}$
return $\mathbf{usk}_1[uid] = (Z, x, y)$

$\mathsf{XSUpdate}^*(t, r, isk_1, (Z, x, y), gpk_1)$

$Q \leftarrow \mathcal{H}(r, t)$
return $(\omega \leftarrow Q^{\frac{1}{\gamma+x}}, Z \cdot \omega)$

$\mathsf{XSSign}^*(I, (\tilde{Z}, x, y), gpk_1, r, t)$

$\rho_1, \rho_2 \leftarrow_\$ \mathbb{Z}_p, T_1 \leftarrow K^{\rho_1}, T_2 \leftarrow \tilde{Z} H^{\rho_1}, T_3 \leftarrow K^{\rho_2}, T_4 \leftarrow \tilde{Z} G^{\rho_2}, z \leftarrow x\rho_1 + y$
$\pi \leftarrow_\$ \mathsf{SPK}\{(x, z, \rho_1, \rho_2) : T_1 = K^{\rho_1} \wedge T_3 = K^{\rho_2} \wedge T_4 T_2^{-1} = G^{\rho_2} H^{-\rho_1} \wedge$
$e(T_2, W) e(T_2, G_2)^x = e(G_1 \cdot \mathcal{H}(r, t), G_2) e(H, W)^{\rho_1} e(H, G_2)^z\}$
return $\Omega = (T_1, T_2, T_3, T_4, \pi)$

$\mathsf{XSVerify}^*(I, r, t, \Omega, gpk_1)$

Parse $\Omega = (T_1, T_2, T_3, T_4, \pi), \tilde{G}_1 = G_1 \cdot \mathcal{H}(r, t)$
return 1 **if** π holds for $T_1, T_2, T_3, T_4, \tilde{G}_1$ **else return** 0

$\mathsf{XSOpen}^*(I, r, t, \Omega, osk, gpk_1)$

if $\mathsf{XSVerify}^*(I, r, t, \Omega, gpk_1) = 0$ **return** $\perp$ **return** $\tilde{Z} \leftarrow T_2 T_1^{-\xi_1}$

Figure 3.5: The algorithms of XS*, our modification to the XS group signature scheme

key, depending on their reputation r at time t. PostItem uses XSSign* to sign as in the original group signature scheme, but with this updated secret key as input. CheckItem uses XSVerify*, which now also takes (r, t) as input, and only outputs 1 if the secret key used to generate this signature has been updated correctly with (r, t).

We also modify the XS scheme by converting it to the static setting, to fit with our reputation system model. We also move to the type–3 pairing setting, as defined in Section 2.3.2, where there is no homomorphism between groups. This means we use the (JoC) q-SDH assumption defined for the type–3 setting.

The XS* signature scheme consists of the algorithms given in Figure 3.5, and the group public parameters gpp_1: the type–3 bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2)$ and hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_1$ that we assume is in the random oracle model. A signature proof of knowledge SPK, as defined in Section 2.6.3, is used as a building block.

$\mathsf{CDLKeyGen}(gpp_2)$

$\alpha, \beta \leftarrow_\$ \mathbb{Z}_p^*, X \leftarrow {G_2}^\alpha, Y \leftarrow {G_2}^\beta$
return $gpk_2 = (X, Y)$, $isk_2 = (\alpha, \beta)$

$\mathsf{CDLJoin}(uid, isk_2, gpk_2)$

$f \leftarrow_\$ \mathbb{Z}_p; F \leftarrow {G_1}^f, r \leftarrow_\$ \mathbb{Z}_p$
$A \leftarrow {G_1}^r, B \leftarrow A^\beta; C \leftarrow (A^\alpha F^{r\alpha\beta}), D \leftarrow (F)^{r\beta}$
$cre \leftarrow (A, B, C, D)$ **return** $\mathbf{usk}_2[uid] \leftarrow (f, cre)$

$\mathsf{CDLSign}(msg, fb, (f, (A, B, C, D)), gpk_2)$

$a \leftarrow_\$ \mathbb{Z}_p, A' \leftarrow A^a, B' \leftarrow B^a, C' \leftarrow C^a, D' \leftarrow D^a$
$J \leftarrow \mathcal{H}(msg)^f, \pi \leftarrow_\$ \mathsf{SPK}\{(f) : D' = B'^f \wedge J = \mathcal{H}(msg)^f\}(msg, fb)$
return $\Phi = (A', B', C', D', J, \pi)$

$\mathsf{CDLVerify}(msg, fb, \Phi, gpk_2)$

Parse $\Phi = (A', B', C', D', J, \pi)$
Verify π with respect to B', D', J, fb, msg
if $A' = 1$ or $J = 1$ **return** 0
if $e(A', Y) \neq e(B', G_2)$ or $e(A'D', X) \neq e(C', G_2)$ **return** 0 **else return** 1

$\mathsf{CDLLink}(msg, (fb_0, \Phi_0), (fb_1, \Phi_1), gpk_2)$

For $b \in \{0, 1\}$ parse $\Phi_b = ({A'}_b, {B'}_b, {C'}_b, {D'}_b, J_b, \pi_b)$
if $\exists b \in \{0, 1\}$ such that $\mathsf{CDLVerify}(msg, fb_b, \Phi_b, gpk_2) = 0$ **return** 0
if $J_0 = J_1$ **return** 1 **else return** 0

Figure 3.6: The algorithms of CDL in the static setting

3.5.2 Direct Anonymous Attestation

The CDL scheme [33], with a merged TPM and host is given in Section 2.8.2. It was proved secure in the full DAA setting, assuming the LSRW [97], Discrete Logarithm (DL), and DDH assumptions, under the state–of–the–art definitions given in [30]. We use the CDL construction in particular because, as shown in Table 1 of [32], it has the lowest estimated running time for signing out of the schemes proved secure under the more recent models. We prioritise efficiency of signing over verification because in reputation systems verification is performed by a server with more computational power.

We provide the CDL scheme in the form used in our reputation system construction in Figure 3.6. The main difference from Section 2.8.2 is that it is in the static setting. The group public parameters gpp_2 are the type–3 bilinear group: $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2)$ and a hash function $\mathcal{H}$, that we assume is in the random oracle model. Again, a signature proof of knowledge SPK, as defined in Section 2.6.3, is used as a building block.

3.5.3 Our RS-GS Construction

In Figure 3.7 we present our RS-GS construction for a reputation system RS, as defined in Section 3.3, derived from the XS* scheme and the CDL scheme.

3.6 Evaluation of our Construction

We first analyse the security of our construction against the conventional attacks against reputation systems, discussed in Section 3.2.2, as well as in the RS model. We then evaluate the efficiency of this construction.

3.6.1 Resilience against Conventional Attacks

The unfair rating attack is mitigated by the soundness of reputation requirement. The on-off attack and whitewashing attack, and collaborative unfair rating attack can be mitigated by design choices detailed in Section 3.2.2. Self–rating attacks could be mitigated by making all users submit the feedback "*" on their own items that could be used to link to self ratings, or be punished by the $\mathcal{CS}$. The central server can also punish authors of items that do not represent a valid transaction.

3.6.2 Security of our Construction

We now show that our construction is a secure RS reputation system as defined in Sections 3.3 and 3.4.

The proofs of traceability and unlinkability of user behaviour are similar to the proofs of traceability/ non-frameability and anonymity for the XS scheme [53]. We have adapted these proofs due to the modification in XS*, and as our model is static (users do not join or leave after the scheme begins).

$\mathsf{Setup}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$

Generate (gpp_1, gpp_2) as in XS, CDL
$(gpk_1, isk_1, osk) \leftarrow\!\!\$\ \mathsf{XSKeyGen^*}(gpp_1), (gpk_2, isk_2) \leftarrow\!\!\$\ \mathsf{CDLKeyGen}(gpp_2)$
$gpk \leftarrow (gpk_1, gpk_2), isk \leftarrow (isk_1, isk_2)$
$\forall uid \in \mathcal{U} \quad \mathbf{usk}_1[uid] \leftarrow\!\!\$\ \mathsf{XSJoin^*}(isk_1, uid, gpk_1), \mathbf{usk}_2[uid] \leftarrow\!\!\$\ \mathsf{CDLJoin}(uid, isk_2, gpk_2)$
$\mathbf{usk}[uid] \leftarrow (\mathbf{usk}_1[uid], \mathbf{usk}_2[uid])$
$\mathcal{L} \leftarrow \emptyset, \mathcal{F} \leftarrow \emptyset, \mathcal{ID} \leftarrow \emptyset, \forall uid \in \mathcal{U}, \mathbf{r}[uid] \leftarrow \hat{r}$ **return** $(gpk, isk, osk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$

$\mathsf{AllocateReputation}(gpk, isk, uid, \mathbf{usk}[uid], t, \mathbf{r}[uid], \mathcal{ID})$

$(\omega, \tilde{Z}) \leftarrow\!\!\$\ \mathsf{XSUpdate^*}(t, \mathbf{r}[uid], \mathbf{usk}_1[uid], isk_1, gpk_1)$
$\mathcal{ID} \leftarrow \mathcal{ID} \cup (uid, \mathbf{r}[uid], t, \tilde{Z})$ **return** $(\omega, \mathbf{r}[uid], \mathcal{ID})$

$\mathsf{PostItem}(gpk, I, (Z, x, y), r, t, \omega)$

if $e(\omega, WG_2^x) \neq e(\mathcal{H}(r,t), G_2)$ **return** $\perp$
$\tilde{Z} \leftarrow Z \cdot \omega, \tilde{usk} \leftarrow (\tilde{Z}, x, y)$ **return** $\Omega \leftarrow\!\!\$\ \mathsf{XSSign^*}(I, \tilde{usk}, gpk_1, r, t)$

$\mathsf{CheckItem}(gpk, I, r, t, \Omega)$

return $\mathsf{XSVerify^*}(I, r, t, \Omega, gpk_1)$

$\mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb)$

return $\Phi \leftarrow\!\!\$\ \mathsf{CDLSign}((I, r, t, \Omega), fb, \mathbf{usk}_2[uid], gpk_2)$

$\mathsf{VerifyFB}(gpk, (I, r, t, \Omega), fb, \Phi)$

return $\mathsf{CDLVerify}((I, r, t, \Omega), fb, \Phi, gpk_2)$

$\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb_0, \Phi_0, fb_1, \Phi_1)$

return $\mathsf{CDLLink}((I, r, t, \Omega), (fb_0, \Phi_0), (fb_1, \Phi_1), gpk_2)$

$\mathsf{ReceiveFB}(gpk, osk, ((I, r, t, \Omega), fb, \Phi), \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$

if $\mathsf{VerifyFB}(gpk, (I, r, t, \Omega), fb, \Phi) = 0$ **return** $\perp$
if $\exists (I, r, t, \Omega), fb', \Phi') \in \mathcal{L}$ s.t. $\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb, \Phi, fb', \Phi') = 1$ **return** $\perp$
$\tilde{Z} \leftarrow \mathsf{XSOpen^*}(I, r, t, \Omega, osk, gpk_1)$, Find $(uid, r, t, \tilde{Z}) \in \mathcal{ID}$, otherwise **return** $\perp$
$\mathbf{r}[uid] \leftarrow \mathsf{Aggr}(fb, uid, \mathcal{F}, \mathbf{r}[uid]), \mathcal{L} \leftarrow ((I, r, t, \Omega), fb, \Phi) \cup \mathcal{L}, \mathcal{F} \leftarrow (uid, fb) \cup \mathcal{F}$ **return** $(\mathbf{r}, \mathcal{L})$

Figure 3.7: Our RS-GS reputation system

3.6.2.1 Correctness

We show our RS-GS construction satisfies correctness. Condition 1 is satisfied because the original XS scheme is correct, and the modification simply replaces G_1 with $G_1\mathcal{H}(r,t)$, in both XSSign* and XSVerify*. Conditions 2 and 3 are satisfied due to the correctness of CDL. Condition 4 is satisfied because the XS scheme is correct, and the modification only replaces Z with $Z(\mathcal{H}(r,t)^{1/(\gamma+x)})$, as the output of XSOpen*. This is stored alongside the user's identity in $\mathcal{ID}$. Condition 5 is satisfied, because ReceiveFB outputs $\perp$ if the feedback input is not valid, or links to other feedback in $\mathcal{L}$.

3.6.2.2 Unforgeability of Reputation

Lemma 3.1 (**Unforgeability of Reputation**)**.** Assuming the random oracle model, the SPK is zero–knowledge and simulation sound extractable, and the q-SDH assumption, our RS-GS construction satisfies unforgeability of reputation as defined in Section 3.4.

Proof. We show that if an adversary $\mathcal{A}'$ exists, such that $\Pr[\mathbf{Exp}^{unforge-rep}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr}) = 1] = \epsilon$, for some $\tau,\mathcal{R},\hat{r},\mathcal{U},\mathsf{Aggr}$ with $|\mathcal{R}|$ polynomial in τ, $n = |\mathcal{U}|$ polynomial in τ, there are l different values of (r,t) queried to the $\mathcal{H}$ oracle, or the ALLREP oracle, and ϵ is non–negligible in τ, then we can can build an adversary $\mathcal{A}$, that solves the q-SDH problem, where $q = n$, in polynomial–time. We describe $\mathcal{A}$ in Figures 3.8 and 3.9. We then describe why the simulation given in Figures 3.8 and 3.9 and the unforgeability of reputation experiment are indistinguishable to $\mathcal{A}$, and how $\mathcal{A}'$ works.

We first show that all inputs that $\mathcal{A}$ provides to $\mathcal{A}'$ are distributed identically to the unforgeability of reputation experiment. As $\mathcal{H}$ is in the random oracle model, we include a $\mathcal{H}$ oracle. We keep track of the inputs and outputs to this oracle in HL

Simulating $(gpk, osk, \mathcal{ID})$**.** $W = T_2^{\gamma} = G_2^{\gamma}$. G_1 is chosen randomly due to μ being chosen randomly and independently. As ν_1 is chosen randomly and independently, H is independent of G_1. $(\xi_1,\xi_2,K,G,\mathcal{ID})$ are chosen as in Setup.

We will use the fact that $\Gamma = G_1^{\gamma}$ and $B_{uid} = G_1^{1/(\gamma+x_{uid})}$ later. This is because $G_1 =$

$\mathsf{USK}(uid)$

if $b = 1$ and $uid = uid^*$ $\mathcal{A}$ aborts **else** $\mathcal{C} \leftarrow \mathcal{C} \cup \{uid\}$ **return** $\mathbf{usk}[uid]$

$\mathsf{POST}(I, uid, r, t, \omega)$

if $b = 1$ and $uid = uid^*$
  if $e(\omega, WG_2^{x_{uid^*}}) \neq e(\mathcal{H}(r,t), G_2)$ **return** $\perp$
  $\rho_1, \rho_2 \leftarrow_\$ \mathbb{Z}_p, T_1 \leftarrow K^{\rho_1}, T_2 \leftarrow Z_{uid^*}\omega H^{\rho_1}, T_3 \leftarrow K^{\rho_2}, T_4 \leftarrow Z_{uid^*}\omega G^{\rho_2}$
  $\tilde{G_1} \leftarrow \mathcal{H}(r,t) \cdot G_1$, Simulate π with $T_1, T_2, T_3, T_4, \tilde{G_1}$
  return $\Omega \leftarrow (T_1, T_2, T_3, T_4, \pi)$
else return $\mathsf{PostItem}(gpk, I, \mathbf{usk}[uid], r, t, \omega)$

$\mathsf{ALLREP}(uid, t, r)$

if $(uid, t, r, \mathsf{out}) \in AR$ **return** $(\mathsf{out}, r, \mathcal{ID})$
else $\mathsf{out}' \leftarrow \mathcal{H}(r,t)$, let $((r,t), \mathsf{out}', \chi, m') \in HL$
if $m' = m^*$ and $b' = 0$ and $uid = uid'$ $\mathcal{A}$ aborts
if $m' = m^*$ and $b' = 0$ and $uid \neq uid'$, $\mathsf{out}'' \leftarrow \prod_{j=0}^{n'-2} (T_1^{\gamma^j})^{\chi\eta_{uid,j}}$
else $\mathsf{out}'' \leftarrow \prod_{j=0}^{n-1} (T_1^{\gamma^j})^{\chi\kappa_{uid,j}}$
$AR \leftarrow AR \cup (uid, t, r, \mathsf{out}''), \mathcal{ID} \leftarrow \mathcal{ID} \cup (uid, r, t, Z_{uid}\mathsf{out}'')$,
return $(\mathsf{out}'', r, \mathcal{ID})$

$\mathcal{H}(\mathsf{in})$

if $\exists(\mathsf{in}, \mathsf{out}, \cdot, \cdot) \in HL$ **return** out **else** $m \leftarrow m + 1, \chi \leftarrow_\$ \mathbb{Z}_p^*$
if $m = m^*$ and $b' = 0, \mathsf{out} \leftarrow \prod_{j=0}^{n'-1} (T_1^{\gamma^j})^{\chi\zeta_j}$
else $\mathsf{out} \leftarrow \prod_{j=0}^{n} (T_1^{\gamma^j})^{\chi\lambda_j}$
$HL \leftarrow (\mathsf{in}, \mathsf{out}, \chi, m) \cup HL$
return out

Figure 3.8: Simulated answers to oracle queries in our unforgeability of reputation proof

$\mathcal{A}(T_1, T_1^{\gamma}, T_1^{\gamma^2}, ..., T_1^{\gamma^q}, T_2, T_2^{\gamma})$

Create Empty List HL, AR, $\quad m \leftarrow 0, m^* \leftarrow_\$ [l], uid' \leftarrow_\$ \mathcal{U}, b \leftarrow_\$ \{0,1\}, b' \leftarrow_\$ \{0,1\}$

if $b = 0 \quad \mathcal{V} \leftarrow \mathcal{U}, n' \leftarrow n$

if $b = 1 \quad uid^* \leftarrow \mathcal{U} \backslash \{uid'\}, \mathcal{V} \leftarrow \mathcal{U} \backslash \{uid^*\}, n' \leftarrow n-1, x_{uid^*} \leftarrow_\$ \mathbb{Z}_p^*, Z_{uid^*} \leftarrow_\$ \mathbb{G}_1$

$\forall uid \in \mathcal{V} \backslash uid', x_{uid}, y_{uid} \leftarrow_\$ \mathbb{Z}_p^*$

$$\mu \leftarrow_\$ \mathbb{Z}_p^*, \text{let } f(X) = \prod_{j \in \mathcal{V} \backslash \{uid'\}} (X + x_{uid}) = \sum_{i=1}^{n'-1} \zeta_i X^i; G_1 \leftarrow \prod_{i=0}^{n'-1} (T_1^{\gamma^i})^{\mu \zeta_i}, \Gamma \leftarrow \prod_{i=0}^{n'-1} (T_1^{\gamma^{(i+1)}})^{\mu \zeta_i}$$

$G_2 \leftarrow T_2, W \leftarrow T_2^{\gamma}, x, \nu_1, \nu_2 \leftarrow_\$ \mathbb{Z}_p^*, H \leftarrow ((\Gamma G_1^x)^{\nu_1} G_1^{-1})^{1/\nu_2}$

$\xi_1 \leftarrow_\$ \mathbb{Z}_p^*, K \leftarrow H^{1/\xi_1}, \xi_2 \leftarrow_\$ \mathbb{Z}_p^*, G \leftarrow K^{\xi_2}, osk \leftarrow (\xi_1, \xi_2), gpk_1 \leftarrow (G_1, K, H, G, G_2, W)$

$$\forall uid \in \mathcal{V} \backslash \{uid'\} \text{ let } f_{uid}(X) = \prod_{\hat{uid} \in \mathcal{V} \backslash \{uid', uid\}} (X + x_{\hat{uid}}) = \sum_{j=0}^{n'-2} \eta_{uid,j} X^j; B_{uid} \leftarrow \prod_{j=0}^{n'-2} (T_1^{\gamma^j})^{\mu \eta_{uid,j}}$$

$$Z_{uid} \leftarrow B_{uid} (B_{uid}^{((x - x_{uid})\nu_1 - 1)/\nu_2} G_1^{\nu_1/\nu_2})^{y_{uid}}, \mathbf{usk}_1[uid] \leftarrow (Z_{uid}, x_{uid}, y_{uid})$$

$y_{uid'} \leftarrow \nu_2, x_{uid'} \leftarrow x, Z_{uid'} \leftarrow G_1^{\nu_1}, \mathbf{usk}_1[uid'] \leftarrow (Z_{uid'}, x_{uid'}, y_{uid'})$

$$\text{Let } g(X) = \prod_{uid \in \mathcal{U}} (X + x_{uid}) = \sum_{j=0}^{n} \lambda_j X^j$$

$$\forall uid \in \mathcal{U}, \text{set } g_{uid}(X) = g(X)/(X + x_{uid}) = \sum_{j=0}^{n-1} \kappa_{uid,j} X^j$$

Finish computing $(\mathbf{usk}_2[uid], gpk_2, \mathcal{ID})$, as in Setup

$(I, r, t, \Omega, fb, \Phi) \leftarrow_\$ \mathcal{A}'^{\mathsf{USK},\mathsf{POST},\mathsf{ALLREP},\mathcal{H}}(gpk, osk, \mathcal{ID})$

Let $\Omega = (T_1, T_2, T_3, T_4, \pi)$

if $\nexists$out such that $((r,t), \mathsf{out}, \chi, m^*) \in HL \quad$ **return** $\perp$

Extract $x^\star, \rho_1^\star, z^\star$ from $\pi, y^\star \leftarrow z^\star - x^\star \rho_1^\star, Q^\star \leftarrow T_2 T_1^{-\xi_1}$

if $x^\star = x_{uid}$ with $uid \in \mathcal{U}$

- **if** $uid \in \mathcal{C} \quad$ **if** $x^\star \neq x_{uid'}$ or $b = 1 \quad$ **return** $\perp$
  - **if** $y^\star = y_{uid'}$, **if** $b' = 1 \quad$ **return** $\perp$ **else** $\quad$ **return** $((Q^\star Z_{uid'}^{-1})^{\mu/\chi}, x^\star)$
  - **if** $y^\star \neq y_{uid'}$, **if** $b' = 0 \quad$ **return** $\perp$ **else** $\omega \leftarrow_\$ \mathsf{ALLREP}(uid', t, r)$
    - **return** $(Q^\star \omega^{-1} G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{(\nu_2 - y^\star)}}, x^\star)$
- **if** $uid \notin \mathcal{C} \quad$ **if** $x^\star \neq x_{uid^*}$ or $b = 0$, or $b' = 0 \quad$ **return** $\perp$
  - $\omega \leftarrow_\$ \mathsf{ALLREP}(uid^*, t, r), Q^\star \leftarrow Q^\star \omega^{-1}$
  - **return** $(Q^\star G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{\nu_2 - y^\star - \nu_1 y^\star (x^\star - x)}}, x^\star)$

else if $b = 1$ or $b' = 0 \quad$ **return** $\perp$ **else return** $(Q^\star G_1^{-\nu_1 y^\star/\nu_2} G_1^{-\chi/\mu})^{\frac{\nu_2}{\nu_2 - y^\star - (\nu_1 y^\star + \chi\nu_2/\mu)(x^\star - x)}}, x^\star)$

Figure 3.9: $\mathcal{A}$ which solves the q-SDH problem, using $\mathcal{A}'$ which breaks unforgeability of reputation for the RS-GS construction

$\prod_{i=0}^{n'-1}(T_1^{\gamma^i})^{\mu\zeta_i} = T_1^{\mu f(\gamma)}$, and so $\Gamma = \prod_{i=0}^{n'-1}(T_1^{\gamma^{(i+1)}})^{\mu\zeta_i} = \prod_{i=0}^{n'-1}(T_1^{\gamma^i})^{\mu\zeta_i\gamma}$ $= G_1^{\gamma}$, and $B_{uid} = \prod_{j=0}^{n'-2}(T_1^{\gamma^j})^{\mu\eta_{uid,j}} = T_1^{\mu f_{uid}(\gamma)} = G_1^{1/(\gamma+x_{uid})}$. This is because $\eta_{uid,j}$ for $j \in [0, n'-2]$ are defined to be the coefficients of f_{uid} in Figure 3.9.

Simulating the USK oracle. The USK oracle is distributed identically to the unforgeability of reputation experiment because, provided the oracle does not abort, if $uid \neq uid'$,

$$Z_{uid} = B_{uid}(B_{uid}^{((x-x_{uid})\nu_1-1)/\nu_2}G_1^{\nu_1/\nu_2})^{y_{uid}} = B_{uid}(B_{uid}^{((x-x_{uid})\nu_1-1)/\nu_2}B_{uid}^{\nu_1(\gamma+x_{uid})/\nu_2})^{y_{uid}}$$

$$= B_{uid}B_{uid}^{y_{uid}(\nu_1(\gamma+x)-1)/\nu_2} = (G_1G_1^{y_{uid}(\nu_1(\gamma+x)-1)/\nu_2})^{1/(\gamma+x_{uid})} = (G_1H^{y_{uid}})^{1/(\gamma+x_{uid})},$$

and $Z_{uid'} = G_1^{\nu_1} = (G_1G_1^{\nu_1(\gamma+x)-1})^{1/(\gamma+x)} = (G_1H^{y_{uid'}})^{1/(\gamma+x_{uid'})}$.

Simulating the POST oracle. If $b = 1$, and $uid = uid^*$ is input to the POST oracle, because Z_{uid^*} was chosen randomly and independently, it is distributed identically to the unforgeability of reputation experiment. The SPK can then be simulated due to the zero–knowledge property. The signature output is then distributed identically to in PostItem. If $b = 0$ or $uid \neq uid^*$, our oracle works in exactly the same way as in the unforgeability of reputation experiment.

Simulating the other oracles. The ALLREP oracle is distributed identically to the unforgeability of reputation experiment, provided the oracle does not abort. This is because with input (uid, t, r), when $m' = m^*$, $uid \neq uid'$, and $b' = 0$,

$$\prod_{j=0}^{n'-2}(T_1^{\gamma^j})^{\chi\eta_{uid,j}} = T_1^{\chi f_{uid}(\gamma)} = T_1^{\chi f(\gamma)/(\gamma+x_{uid})} = \mathcal{H}(r,t)^{1/(\gamma+x_{uid})},$$

else $\prod_{j=0}^{n-1}(T_1^{\gamma^j})^{\chi\kappa_{uid,j}} = T_1^{\chi g_{uid}(\gamma)} = T_1^{\chi g(\gamma)/(\gamma+x_{uid})} = \mathcal{H}(r,t)^{1/(\gamma+x_{uid})}$. The hash oracle is distributed identically to the random oracle model, because χ is chosen randomly and independently each time.

Reduction to q-SDH. If $\mathcal{A}'$ is successful and outputs (I, r, t, Ω) then there exists out such that $((r,t), \mathsf{out}, m) \in HL$, because for $\mathcal{A}'$ to have output a valid signature for (r,t), this must have been queried to the $\mathcal{H}$ oracle. Assume $m = m^*$, with probability

$1/l$, then $\mathcal{A}$ does not abort at this stage. As (I, r, t, Ω) was not output by POST, we can extract $(Q^\star, x^\star, y^\star)$ such that $Q^\star = (G_1, H^{y^\star}\mathsf{out})^{1/(x^\star+\gamma)}$, because Ω is a valid signature.

We now consider all possible cases for a successful $\mathcal{A}'$, and provide reductions to the q-SDH problem for each case. The three cases are as follows: firstly $x^\star$ corresponds to the key of a corrupted user, ie $x^\star = x_{uid}$ for $uid \in \mathcal{C}$; secondly $x^\star$ corresponds to the key of an uncorrupted user, ie $x^\star = x_{uid}$ for $uid \in \mathcal{U} \setminus \mathcal{C}$; and finally $x^\star$ does not correspond to the key of any user, ie $x^\star \neq x_{uid}$ for all $uid \in \mathcal{U}$. The first cases is split into two sub cases: firstly $y^\star = y_{uid}$ where $x^\star = x_{uid}$ and secondly $y^\star \neq y_{uid}$ where $x^\star = x_{uid}$.

When $x^\star$ corresponds to the key of a corrupted user. When $x^\star = x_{uid}$ for $uid \in \mathcal{C}$, we assume $b = 0$ and $uid = uid'$, which occurs with probability $1/2n$. As $b = 0$, USK will not abort.

We first note that as $\mathcal{A}'$ is successful, if $x^\star = x_{uid'}$ and $y^\star = y_{uid'}$, then as $\mathsf{ReceiveFB}$ would successfully trace user uid' then (uid', r, t) has not been queried to ALLREP. Therefore ALLREP will not abort.

We assume $b' = 0$, which occurs with probability $1/2$ and therefore, $\mathsf{out} = \prod_{j=0}^{n'-1}(T_1^{\gamma^j})^{\chi\zeta_j} = G_1^{\chi/\mu}$.

Then $Q^\star Z_{uid'}^{-1} = \mathsf{out}^{1/(x_{uid'}+\gamma)}$. Therefore $(\mathsf{out}^{1/(\gamma+x_{uid'})})^{\mu/\chi} = G_1^{1/(\gamma+x_{uid'})}$. Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20] . Therefore, in this case, assuming $\mathcal{A}'$ was successful, $\mathcal{A}$ succeeds with probability $1/4ln$

If $y^\star \neq y_{uid'}$, we assume $b' = 1$ therefore $\mathcal{A}$ will not abort during ALLREP, and we can query $\mathsf{ALLREP}(uid', t, r)$. As $\nu_2 = y_{uid'} \neq y^\star$, then $(\nu_2 - y^\star) \neq 0$. Note that $x = x^\star$.

Because $Q^\star\omega^{-1} = (G_1 H^{y^\star})^{1/(x^\star+\gamma)}$, then

$$(Q^\star\omega^{-1}G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{\nu_2-y^\star}} = (G_1H^{y^\star})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2-y^\star)}} G_1^{\frac{-\nu_1 y^\star}{\nu_2-y^\star}}$$

$$= (G_1 G_1^{y^\star(\nu_1(\gamma+x)-1)/\nu_2})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2-y^\star)}} G_1^{\frac{-\nu_1 y^\star}{\nu_2-y^\star}} = G_1^{\frac{y^\star\nu_1(\gamma+x)-y^\star+\nu_2-y^\star\nu_1(\gamma+x^\star)}{(\gamma+x^\star)(\nu_2-y^\star)}} = G_1^{1/(\gamma+x^\star)}.$$

Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20]. Therefore, in this case,

assuming $\mathcal{A}'$ was successful, $\mathcal{A}$ succeeds with probability $1/4ln$.

When $x^\star$ corresponds to the key of an uncorrupted user. We now consider the case that $uid \notin \mathcal{C}$. In this case, we assume $x^\star = x_{uid^*}$, $b = 1$, and $b' = 1$, which occurs with probability $1/4n$. Then as $b' = 1$, $\mathcal{A}$ will not abort during ALLREP, and as uid^* is not corrupted, $\mathcal{A}$ will not abort during USK. If $\nu_2 - y^\star - \nu_1 y^\star(x^\star - x) = 0$, then $\nu_1 = \frac{\nu_2 - y^\star}{y^\star(x^\star - x)}$. If $y^\star = 0$ then $\nu_2 = 0$, which is not possible. Therefore the adversary can obtain ν_1 and so break the discrete logarithm problem, which is implied by the q-SDH problem.

Because $Q^\star \omega^{-1} = (G_1 H^{y^\star})^{1/(x^\star+\gamma)}$, then

$$(Q^\star \omega^{-1} G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{\nu_2 - y^\star - \nu_1 y^\star(x^\star - x)}} = (G_1 H^{y^\star})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}} G_1^{\frac{-\nu_1 y^\star}{\nu_2 - y^\star - \nu_1 y^\star(x^\star - x)}}$$

$$= (G_1 G_1^{y^\star(\nu_1(\gamma+x)-1)/\nu_2})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}} G_1^{\frac{-\nu_1 y^\star}{\nu_2 - y^\star - \nu_1 y^\star(x^\star - x)}}$$

$$= G_1^{\frac{y^\star \nu_1(\gamma+x) - y^\star + \nu_2 - y^\star \nu_1(\gamma+x^\star)}{(\gamma+x^\star)(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}} = G_1^{1/(\gamma+x^\star)}.$$

Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20]. Therefore, in this case, assuming $\mathcal{A}'$ was successful, $\mathcal{A}$ succeeds with probability $1/(4ln)$.

When $x^\star$ does not correspond to a user's key. If $x^\star \notin \{x_{uid} : uid \in \mathcal{U}\}$, we assume $b' = 1$ and $b = 0$, which occurs with probability $1/4$, and therefore $\mathcal{A}$ will not abort. If $\nu_2 - y^\star - (\nu_1 y^\star + \chi\nu_2/\mu)(x^\star - x) = 0$, then $\chi = \frac{\mu(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}{\nu_2(x^\star - x)}$. Therefore the adversary can obtain χ and so break the discrete logarithm problem, which is implied by the q-SDH problem.

Due to the fact that ,

$$\mathsf{out}^{1/(\gamma+x^\star)} = \prod_{j=0}^{n} (T_1^{\gamma^j})^{\chi\lambda_j}{}^{1/(\gamma+x^\star)} = T_1^{\chi g(\gamma)/(\gamma+x^\star)} = G_1^{\frac{\chi(\gamma+x)}{\mu(\gamma+x^\star)}}$$

$$= \Gamma^{\chi/\mu(\gamma+x^\star)} G_1^{\chi x/\mu(\gamma+x^\star)} = (\Gamma^{\chi/\mu} G_1^{\chi x^\star/\mu})^{1/(\gamma+x^\star))} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}} = G_1^{\chi/\mu} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}},$$

and $Q^\star = (G_1 H^{y^\star} \mathsf{out})^{1/(\gamma+x^\star)}$,

and

$$(G_1 H^{y^\star})^{1/(\gamma+x^\star)} = (G_1 G_1^{y^\star(\nu_1(\gamma+x)-1)/\nu_2})^{1/(\gamma+x^\star)}$$

$$= G_1^{\frac{y^\star\nu_1 x^\star}{\nu_2(\gamma+x^\star)}} G_1^{\frac{y^\star\nu_1\gamma}{\nu_2(\gamma+x^\star)}} (G_1 G_1^{y^\star(\nu_1(x-x^\star)-1)/\nu_2})^{1/(\gamma+x^\star)} = G_1^{y^\star\nu_1/\nu_2} G_1^{\frac{\nu_2+y^\star(\nu_1(x-x^\star)-1)}{\nu_2(\gamma+x^\star)}},$$

we have that

$$(Q^\star G_1^{-\nu_1 y^\star/\nu_2} G_1^{-\chi/\mu})^{\frac{\nu_2}{\nu_2-y^\star-(\nu_1 y^\star+\chi\nu_2/\mu)(x^\star-x)}}$$

$$= (G_1^{y^\star\nu_1/\nu_2} G_1^{\frac{\nu_2+y^\star(\nu_1(x-x^\star)-1)}{\nu_2(\gamma+x^\star)}} G_1^{-\nu_1 y^\star/\nu_2} G_1^{\chi/\mu} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}} G_1^{-\chi/\mu})^{\frac{\nu_2}{\nu_2-y^\star-(\nu_1 y^\star+\chi\nu_2/\mu)(x^\star-x)}}$$

$$= (G_1^{\frac{\nu_2+y^\star(\nu_1(x-x^\star)-1)}{\nu_2(\gamma+x^\star)}} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}})^{\frac{\nu_2}{\nu_2-y^\star-(\nu_1 y^\star+\chi\nu_2/\mu)(x^\star-x)}} = G_1^{1/(\gamma+x^\star)}.$$

Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20] . Therefore, in this case, assuming $\mathcal{A}'$ was successful, $\mathcal{A}$ succeeds with probability $1/4l$.

Therefore $\mathcal{A}$ solves the q-SDH problem with probability at least $\frac{\epsilon}{4ln}$.

□

3.6.2.3 Traceability

Lemma 3.2 (**Traceability**)**.** Assuming the random oracle model, the SPK is zero–knowledge and simulation sound extractable, and the q-SDH assumption, our RS-GS construction satisfies traceability as defined in Section 3.4.

Our proof of traceability is very similar to our proof of unforgeability of reputation. We therefore present this proof in Appendix A.

3.6.2.4 Unlinkability of User Behaviour

Lemma 3.3 (**Unlinkability of User Behaviour**)**.** Assuming the DDH assumption and the SPK is zero–knowledge, our RS-GS construction satisfies unlinkability of user behaviour as defined in Section 3.4.

Proof. We show that if an adversary $\mathcal{A}'$ exists, such that $|\Pr[\mathbf{Exp}^{anon-ub-0}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr})$

$\mathsf{RECEIVEFB}((I, r, t, \Omega), fb, \Phi, \mathcal{ID})$

if $\mathsf{VerifyFB}(gpk, (I, r, t, \Omega), fb, \Phi) = 0$ **return** $\perp$
if $\exists (I, r, t, \Omega), fb', \Phi') \in \mathcal{L}$ such that $\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb, \Phi, fb', \Phi') = 1$
return $\perp$
if $d = 0$ $\tilde{Z} \leftarrow T_2 T_1^{-\xi_1}$, **if** $d = 1$ $\tilde{Z} \leftarrow T_4 T_3^{-\xi_2}$
Find $(uid, r, t, \tilde{Z}) \in \mathcal{ID}$, otherwise **return** $\perp$
$\mathbf{r}[uid] \leftarrow \mathsf{Aggr}(fb, uid, \mathcal{F}, \mathbf{r}[uid]), \mathcal{L} \leftarrow ((I, r, t, \Omega), fb, \Phi) \cup \mathcal{L}, \mathcal{F} \leftarrow (uid, fb) \cup \mathcal{F}$
return $(\mathbf{r}, \mathcal{L}, \mathcal{F})$

$\mathcal{A}(K, T, U, V)$

$b, d, d' \leftarrow_\$ \{0, 1\}$
if $d = 0$ $\xi_1 \leftarrow_\$ \mathbb{Z}_p^*, H \leftarrow K^{\xi_1}, G \leftarrow T$, **if** $d = 1$ $\xi_2 \leftarrow_\$ \mathbb{Z}_p^*, H \leftarrow T, G \leftarrow K^{\xi_2}$
Other than (K, H, G) generate $(gpk, isk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$ as in Setup
$(\mathsf{st}, uid_0, uid_1, I, r, t, \mathcal{ID}) \leftarrow_\$ \mathcal{A}'^{\mathsf{RECEIVEFB}}(\mathsf{choose}, gpk, isk, \mathbf{usk}, \mathbf{r}, \mathcal{L}, \mathcal{F}, \mathcal{ID})$
$\mathsf{out} \leftarrow \mathcal{H}(r, t), \tilde{G}_1 \leftarrow G_1 \mathsf{out}, \omega_0 \leftarrow \mathsf{out}^{1/(\gamma + x_{uid_0})}, \omega_1 \leftarrow \mathsf{out}^{1/(\gamma + x_{uid_1})}$
$\mathcal{ID} \leftarrow \mathcal{ID} \cup \{(uid_0, r, t, Z_{uid_0}\omega_0), (uid_1, r, t, Z_{uid_1}\omega_1)\}$
if $d = 0$ $\rho_1 \leftarrow_\$ \mathbb{Z}_p, T_1 \leftarrow K^{\rho_1}, T_2 \leftarrow Z_{uid_{d'}}\omega_{d'} H^{\rho_1}, T_3 \leftarrow U, T_4 \leftarrow Z_{uid_b}\omega_b V$
if $d = 1$ $\rho_2 \leftarrow_\$ \mathbb{Z}_p, T_1 \leftarrow U, T_2 \leftarrow Z_{uid_b}\omega_b V, T_3 \leftarrow K^{\rho_2}, T_4 \leftarrow Z_{uid_{d'}}\omega_{d'} G^{\rho_2}$
Simulate π with $T_1, T_2, T_3, T_4, \tilde{G}_1$, $\Omega \leftarrow (T_1, T_2, T_3, T_4, \pi)$
$b' \leftarrow_\$ \mathcal{A}'^{\mathsf{RECEIVEFB}}(\mathsf{guess}, \mathsf{st}, \Omega, \mathcal{ID})$
if $((I, r, t, \Omega), \cdot)$ queried to the $\mathsf{RECEIVEFB}$ oracle $b' \leftarrow 0$
if $b' = b$ **return** 1 **else return** 0

Figure 3.10: $\mathcal{A}$ which distinguishes between DDH tuples in $\mathbb{G}_1$, using $\mathcal{A}'$ which breaks unlinkability of user behaviour for our $\mathsf{RS\text{-}GS}$ construction

$= 1] - \Pr[\mathbf{Exp}^{anon-ub-1}_{\mathcal{A}', \mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1]| = \epsilon$, for some $\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}$ with $|\mathcal{R}|$ polynomial in τ, $n = |\mathcal{U}|$ polynomial in τ, and ϵ is non–negligible in τ, then we can can build an adversary $\mathcal{A}$ that breaks the DDH assumption with non-negligible probability. We describe $\mathcal{A}$ in Figure 3.10. We then describe why the simulation given in Figure 3.10 and the unlinkability of user behaviour experiment are indistinguishable to $\mathcal{A}$ if a DDH tuple is input, and $\mathcal{A}$ chooses $d' = b$. We then show that, otherwise, $\mathcal{A}$ guesses correctly with probability $1/2$, and therefore $\mathcal{A}$ successfully distinguishes DDH tuples.

Simulating the inputs to $\mathcal{A}'$. Assuming $\mathcal{A}$ was input a DDH tuple and chooses $d' = b$, all inputs to $\mathcal{A}'$ are distributed identically to the unlinkability of user behaviour experiment. In the choosing phase, the only difference to the experiment is in choosing (K, H, G), which are still distributed identically. In the guessing phase, as $d' = b$ and (K, T, U, V) is a DDH tuple,the input Ω to $\mathcal{A}'$ is also distributed identically to the unlinkability of

user behaviour Experiment, and $\mathcal{ID}$ is modified in the same way as the experiment. The cca-anonymity property of the XS group signature scheme [53] is used to simulate answers to the RECEIVEFB oracle. Since $T_4 T_3^{\xi_2} = T_2 T_1^{-\xi_1}$ for a valid signature, to open signatures it is only necessary to know ξ_1 or ξ_2.

Reduction to the DDH problem. Assuming $\mathcal{A}$ was input a DDH tuple, and chooses $d' \neq b$, both $Z_{uid_0}\omega_0$ and $Z_{uid_1}\omega_1$ are encrypted. If the adversary $\mathcal{A}'$ has queried the RECEIVEFB oracle with (I, r, t, Ω), then $\mathcal{A}$ outputs 1 with probability 1/2. Otherwise, the adversary $\mathcal{A}'$ has not queried the RECEIVEFB oracle with (I, r, t, Ω) to obtain d', and therefore $\mathcal{A}'$ has no advantage in guessing b. Therefore $\mathcal{A}$ outputs 1 with probability 1/2.

On input a DDH tuple, the probability that $\mathcal{A}$ outputs 1 is:

$$1/4(\Pr[\mathbf{Exp}^{anon-ub-0}_{\mathcal{A}',\text{RS-GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \text{Aggr}) = 0] + \Pr[\mathbf{Exp}^{anon-ub-1}_{\mathcal{A}',\text{RS-GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \text{Aggr}) = 1]) + 1/4$$

$$= 1/4(1 + \Pr[\mathbf{Exp}^{anon-ub-1}_{\mathcal{A}',\text{RS-GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \text{Aggr}) = 1] - \Pr[\mathbf{Exp}^{anon-ub-0}_{\mathcal{A}',\text{RS-GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \text{Aggr}) = 1])$$

$$+1/4 = 1/4\epsilon + 1/2.$$

If the input to $\mathcal{A}$ is not a DDH tuple, then Ω is independent of b, therefore $\mathcal{A}'$ has no advantage in guessing b. Therefore $\mathcal{A}$ outputs 1 with probability 1/2.

Therefore $\mathcal{A}$ has an $\epsilon/4$ advantage in distinguishing DDH tuples.

□

3.6.2.5 Soundness of Reputation, Anonymity of Feedback and Non-frameability

Lemma 3.4 (**Soundness of Reputation**)**.** Assuming that CL signatures are existentially unforgeable under the chosen–message attack, the SPK is zero–knowledge and simulation sound extractable, and the random oracle model, our RS-GS construction satisfies soundness of reputation as defined in Section 3.4.

Lemma 3.5 (**Anonymity of Feedback**)**.** Assuming the DDH assumption in $\mathbb{G}_1$, the SPK is zero–knowledge, and the random oracle model, our RS-GS construction satisfies

anonymity of feedback as defined in Section 3.4.

Lemma 3.6 (**Non–frameability**)**.** Assuming the DL assumption in $\mathbb{G}_1$, the SPK is zero–knowledge and simulation sound extractable, and the random oracle model, our RS-GS construction satisfies non–frameability as defined in Section 3.4.

The proofs of Lemmas 4, 5, 6 are similar to the simulation based proof of security of CDL [33]. It is clear, due to the similarity of the security requirements for DAA schemes [33] and the security requirements of soundness of reputation, anonymity of feedback and non-frameability, that a reputation system that uses the CDL scheme will satisfy these requirements. We present the proofs in Appendix A for completeness.

Soundness of reputation ensures that an adversary cannot submit multiple feedback on the same item. This corresponds to the property for DAA that a user should not be able to produce two signatures with the same basename that do not link. Anonymity of feedback ensures that, provided a user has not cheated by giving multiple feedback on the same item, all feedback is unlinkable. This corresponds to the user-controlled linkability property for DAA schemes that signatures with different basenames should be unlinkable. Non–frameability ensures that an adversary cannot produce feedback that links to another user's feedback, without corrupting this user. This corresponds to the property for DAA that an adversary should not be able to forge a signature that links to a user's signature, without corrupting this user.

3.7 Instantiation of SPK and Efficiency

3.7.1 Instantiation of SPKs

We have two non-interactive zero-knowledge proofs of knowledge in our scheme, in XSSign* and CDLSign. Given a proof of discrete logarithms in this notation, as discussed in Section 2.6.3, it is straightforward to derive an actual protocol implementing the proof [35]. For transforming interactive into non-interactive zero-knowledge proofs we rely on the Fiat-Shamir heuristic that ensures security in the random oracle model.

Due to this, we can now state Corollary 3.1.

Corollary 3.1. The construction presented in Section 3.5.3, with the SPK instantiated as described, is a secure RS as defined in Sections 3.3 and 3.4 under the DDH, q-SDH, and LRSW assumptions in the random oracle model.

3.7.2 Computational Cost

We focus on PostItem, CheckItem, and SendFB, because these are performed by users with less computational power. We note that ReceiveFeedback only needs to check all feedback for the same item to ensure soundness of reputation, not all feedback. SendFB requires 7 exponentiations in $\mathbb{G}_1$, and 2 hash computations, which is a low computational cost.

There is an extra cost, compared to the existing work [18, 58], required to achieve unlinkability of user behaviour in the PostItem and CheckItem algorithms. We note that in [18, 58], whenever a user posts an item they must receive a new secret key from the managing authority, which is not required by our reputation system. When joining, the user pre–computes $e(H, W)$ and $e(H, G_2)$. When a user receives a token ω, for reputation r and time t, they must pre–compute $e(\omega, WG_2^x)$, $e(\mathcal{H}(r, t), G_2)$, and $e(\omega Z, G_2)$. Assuming the pre–computation, PostItem requires 8 exponentiations in $\mathbb{G}_1$, 3 exponentiations in $\mathbb{G}_3$, and a hash computation. CheckItem requires 2 computations of e, 10 exponentiations in $\mathbb{G}_1$, 2 exponentiations in $\mathbb{G}_2$, 2 exponentiations in $\mathbb{G}_3$, and 2 hash computations.

3.7.3 Communication Overhead

Using updated parameters for curves that provide 128–bit security [119] and point compression, the XS* signature Ω has length 432 bytes and the CDL signature Φ has length 336 bytes. Therefore the communication overhead when sending feedback with our construction is 768 bytes, compared to 624 bytes in [18] using the same curves. This is a relatively small increase given the additional security of unlinkability for user behaviour achieved. Our communication overhead compares well to [58] where signatures have length $\mathcal{O}(\tau \log(n))$, as shown in [93], compared to our signatures of length $\mathcal{O}(\tau)$.

3.8 Summary

We have introduced and formally defined a new security model for centralised reputation systems, where user behaviour is unlinkable. This represents a shift from previous models which aims at more accurately capturing the real-world requirements of reputation systems used by many on a daily basis.

We have provided a concrete construction which satisfies the new security requirements with a low additional efficiency cost.

Chapter 4

Group Signatures with Selective Linkability

Contents

4.1 Introduction

This chapter introduces group signatures with selective linkability, which allow signatures to be linked in a controlled and flexible way. Signatures are unlinkable by default, but can be obliviously and non–transitively linked by the converter, a trusted entity. We first introduce a formal security model for this primitive and then provide a construction that provably satisfies this model.

4.1.1 Motivation and Background

Group signatures are highly suited whenever data is collected that needs to be authenticated while, at the same time, the privacy of the data sources must be respected and preserved. In particular when data is collected from users, the protection of their privacy is of crucial importance and has seen increased attention due to the recently introduced General Data Protection Regulation (GDPR) [2], Europe's new privacy regulation. In fact, the GDPR creates strong incentives for data collectors to thoroughly protect users' data and implement the principle of data minimization, as data breaches are fined with up to 4% of an enterprise's annual turnover.

When aiming to implement such techniques for privacy and data protection, one needs to find a good balance with utility though since data gets collected in order to be analysed and to generate new insights. For these processes it is usually necessary to know the correlation among different data events, as they reveal a crucial part of the information. For instance, when a group of users measure and upload their blood pressure via wearable activity trackers, several high value measurements are not critical when they are distributed over many participants, but might be alarming when originating from a single user.

Often the exact purpose of the data might not be clear at the point of data collection. In fact, given the rapid advancements in machine learning and ubiquitously available and cheap storage, data collectors tend to gather large amounts of data at first, and will only use small subsets for particular applications as they arise. A well known example is the Google Street View cars that inadvertently recorded public Wi-Fi data like SSID information, which later was used to improve Google's location services.

Ideally, the data should be collected and stored in authenticated and unlinkable form. Only the particular subsets that are later needed should be correlated in a controlled and flexible manner.

4.1.2 Linkability in Group Signatures

As discussed in Section 2.7.3, to address the tension between privacy and utility, group signatures often have built-in measures that control linkability of otherwise anonymously

authenticated information. Interestingly, despite the long line of work on this subject, none of the solutions provides the functionality to cater for the flexibility needed in practice. They either recover linkability in a privacy-invasive way or offer control only in a static manner.

Group Signatures with Opening. Standard group signatures allow an entity with an opening secret key to recover the signer's identity. Originally, the opening was intended to prevent abuse of anonymity, and only meant to be used in extreme situations. Clearly, the opening capability can also be leveraged to determine the linkability of various data events, but at a high cost for privacy: every request for linkability will recover the full identity of the signer, and the central group manager learns the (signed) data of the data collectors and their correlation.

Group Signatures with Controlled Linkability. A more suitable solution are group signatures with *controlled linkability* [79, 80, 120]. This is much better than revealing the identity of the user, but still relies on a *fully trusted* entity that will learn the collectors' signed data. Further, this approach does not scale well for applications where a data collector is interested in the correlations within a large data set. To link a data set of n signed entries, each pair of signatures would have to be compared, which would require $n(n-1)/2$ requests to the linking authority. Another related concept are *traceable* group signatures [83] where a dedicated entity can generate a tracing trapdoor for each user which allows for the tracing of this user's signatures. This approach is not suitable for our use case of controlled data linkage either, as it requires knowledge of the users' identities behind the anonymous group signatures or trapdoors for *all* users, and also needs every signature to be tested for every trapdoor.

Group Signatures with User-Controlled Linkability. Finally, schemes with *user-controlled linkability* exist, such as in direct anonymous attestation (DAA) [29], described in Chapter 2. In contrast to solutions with opening or linking authorities, the linkability here can be publicly verified: a signature in such schemes contains a *pseudonym* that is deterministically derived from the user's secret key and the basename. Thus, the user re-uses the same pseudonym whenever they want to be linkable. On the downside, this linkage is immediate and static. That is, the users have to choose at the beginning whether

they want to disclose their data in a fully unlinkable manner, or linked with respect to a context-specific pseudonym. There is no option to selectively correlate the data after it has been disclosed. Therefore, users or rather the data collectors allowing the use of such protocols, will hesitate to choose the option of unlinkability, as they fear losing too much information by the irreversible decorrelation.

4.1.3 Our Contribution

In this chapter we overcome the aforementioned limitations by introducing a new type of group signature scheme that allows for flexible and selective linkability. We achieve that functionality by combining ideas from the different approaches discussed previously. Group signatures are associated with pseudonyms, but pseudonyms are *unlinkable by default*. Only when needed, a set of signatures – or rather the pseudonyms – can be linked in an efficient manner through a central entity, the *converter*. The converter receives a batch of pseudonymous data and transforms this into a consistent representation, meaning that all pseudonyms stemming from the same user will be converted into the same value. To preserve the privacy of the users and their data, the converter correlates the data in a fully *blind* way, i.e., they do not learn anything about the pseudonyms they transform. We call this new type of group signature scheme *convertibly linkable (group) signatures*, i.e. CLS.

Security and Privacy for CLS. A crucial property that we want from pseudonym conversions is that they establish linkability only strictly within the queried data, i.e., linked pseudonyms from different queries should not be transitive. Otherwise, different re-linked data sets with overlapping input data could be pieced together, thereby gradually eroding the user's privacy. Aiming for such *non-transitivity* has an immediate impact on the overall setting: we need to channel both, the pseudonyms and messages, blindly through the converter, as transforming pseudonyms without the messages would require linkability between the in- and outputs of the conversion query, which in turn allows outputs from different queries to be linked.

We formally define the security of CLS through a number of security games, strongly inspired by the existing work on group signatures and DAA [12, 14, 33]. That is, we want signatures to be fully *anonymous* and unlinkable bearing in mind the information that is revealed through the selective linkability. We discuss that the classic anonymity notion

adapted to our setting will not suffice, as it cannot guarantee the desired *non-transitivity*. In fact, capturing the achievable privacy and non-transitivity property in the presence of adaptive conversion queries was one of the core challenges in this chapter, and we formalise this property through a simulation-based definition. If the converter is corrupt, then unlinkability of signatures no longer holds, but the adversary should neither be able to trace signatures to a particular user, nor harm the obliviousness of queries, which is captured in the *conversion blindness* and *join anonymity* properties. The guarantees in terms of unforgeability are captured through the *non-frameability* and *traceability* requirements. The former says that corrupt users should not be able to impersonate honest users, and the latter guarantees that the power of an adversary should be bounded by the number of corrupt users they control.

From a corruption point of view, we assume the data collector to be at most *honest–but–curious* towards the converter, i.e., even a corrupt data collector will only query pseudonym-message pairs that it has received along with a valid signature. We consider this a reasonable assumption, as data collectors that will use such a CLS scheme do so in order to implement the principle of data minimization on their own premises, and do not have an incentive to cheat themselves. We will relax this assumption in Chapter 6, although this will come with an efficiency cost.

Efficient Instantiation. We propose an efficient construction of such CLS schemes, following the classical sign-and-encrypt paradigm that underlies most group signatures. Roughly, we use BBS+ signatures [9] for attesting group membership, i.e., a user will blindly receive a BBS+ signature from the group issuer on a secret key y chosen afresh by the user. To sign a message m on behalf of the group, the user computes a signature-proof-of-knowledge (SPK) for m where they prove knowledge of such an issuer's signature on its secret key and also encrypts its user key (or rather its "public key" version h^y, where h is a public parameter of the scheme), under the converter's public key. The ciphertext that encrypts h^y serves as the pseudonym μ.

When the converter is asked to recover the correlations for a set of k pseudonym-message pairs $(\mu_1, m_1), \ldots (\mu_k, m_k)$, it blindly decrypts each pseudonym and raises the result to the power of r, which is chosen fresh for every conversion query but used consistently within. That is, all pseudonyms belonging to the same user will be mapped to the same

query-specific DDH tuple h^{yr} which allows for linkage of data within the query, but guarantees that converted pseudonyms remain unlinkable across queries. To achieve obliviousness and non-transitivity of conversions, we encrypt all pseudonyms and messages with a re-randomisable (homomorphic) encryption scheme under the blinding key of the data collector. The re-randomisation is applied by the converter before they return the transformed values, which ensures that the data collector cannot link the original and the converted pseudonyms by any cryptographic value. Clearly, if the associated messages are unique, then the data collector can link in- and outputs anyway, but our scheme should not introduce any additional linkage. Given that the pseudonyms are encryptions under the converter's public key, we need to add the second layer of encryption in a way that it does not interfere with the capabilities of the inner ciphertext. Using a nested form of ElGamal encryption [60] gives us these properties as well as the needed re-randomisability.

Finally, we prove that our instantiation satisfies the desired security and privacy requirements under the DDH, q-SDH and DCR assumptions in the random oracle model. Our construction relies on type–3 pairings and performs most of the work in $\mathbb{G}_1$ which comes with significant efficiency benefits. In fact, we show that our construction is reasonably efficient considering the increased flexibility when establishing the linkability in such a selective and controlled manner.

4.1.4 Other Related Work

A number of results exist that establish convertible pseudonyms in the setting of distributed databases and have inspired our work. The data is created and maintained in a distributed manner. For privacy, related data is stored under different, database-specific pseudonyms that are seemingly unlinkable and can only be correlated by a central entity that controls the data flow. While the initial approach by Galindo and Verheul [68] required the converter to be a trusted third party, Camenisch and Lehmann [36, 37] later showed how the converter can operate in an oblivious manner. However, none of these solutions supports *authenticated* data collection and [68] and [36] even let the (trusted) converter establish all pseudonyms. The pseudonym system in [37] bootstraps pseudonyms in a blind way from a user secret, but for every new pseudonym that requires the user, converter and targeted data base to engage in an interactive protocol. Clearly, this is not practical for a setting where users frequently want to upload data. Further, all schemes

re-use the same pseudonym for a user within a database, whereas our solution creates fresh and unlinkable pseudonyms for every new data item.

4.2 Definition and Security Model for CLS

In this section we first introduce the syntax and generic functionality of CLS and then present the desirable security and privacy properties for such schemes.

The following entities are involved in a CLS scheme: an *issuer* $\mathcal{I}$, a set of users $\mathcal{U} = \{uid_i\}$, a *verifier* $\mathcal{V}$ and a *converter* $\mathcal{C}$. The issuer $\mathcal{I}$ is the central entity that allows users to join the group. Once joined, a user can then sign on behalf of the group in a pseudonymous way. That is, a verifier $\mathcal{V}$ can test the validity of a signature with respect to the group's public key but does not learn any information about the particular user that created the signature. Most importantly, we want the pseudonymously signed data to be linkable in a controlled yet blind manner. Such selected linkability can be requested through the converter $\mathcal{C}$ that can blindly transform tuples of pseudonym-message pairs into a consistent representation.

4.2.1 Syntax of CLS

Our notation closely follows the definitional framework for dynamic group signatures given in [14], although there are some differences due to the different setting. Instead of an opener, we have a converter who outputs a linked representation of the signatures input. This means we no longer have need of a registration table for joined users. As we assume the converter is honest–but-curious, we no longer need the equivalent of a judge algorithm.

We also weaken the anonymity requirement so that forward anonymity is no longer provided. It seems difficult to achieve this whilst also ensuring the non–transitivity of conversions. Now if the adversary is allowed to query the USK oracle, the adversary could have instead created that user with the issuer secret key. Therefore there is no longer any need for the USK oracle.

We stress that our algorithms (and security notions) are flexible enough to cover settings

where multiple verifiers and converters exist. For the sake of simplicity, however, we focus on the setting where there is only one entity each.

Definition 4.1 (CLS)**.** A convertibly linkable group signature CLS scheme consists of the following algorithms:

Setup and Key Generation. We model key generation individually per party, and refer to $(\mathsf{param}, ipk, cpk)$ as the group public key gpk.

CLS.Setup$(1^\tau) \to$ **param:** on input the security parameter 1^τ, outputs param, the public parameters for the scheme.

CLS.IKGen(**param**) $\to (ipk, isk)$**:** performed by the issuer $\mathcal{I}$; outputs the issuer secret key isk, and the issuing public key ipk.

CLS.CKGen(**param**) $\to (cpk, csk)$**:** performed by the converter $\mathcal{C}$; outputs the converter secret key csk, and the converter public key cpk.

CLS.BKGen(**param**) $\to (bpk, bsk)$**:** performed by the verifier $\mathcal{V}$[1]; outputs a blinding secret key bsk, and blinding public key bpk. As the key is only used for blinding purposes, (bpk, bsk) can be ephemeral. We write $\mathcal{BK}$ as the public key space induced by CLS.BKGen.

Join, Sign and Verify. As in standard dynamic group signatures, we have a dedicated join procedure that a user has to complete with the issuer. All users that have successfully joined the group can then create pseudonymous signatures on behalf of the group, i.e., that verify with respect to the group public key gpk. For ease of expression we treat the pseudonym μ as a dedicated part of the signature.

$\langle$**CLS.Join**(gpk), **CLS.Issue**$(isk, gpk)\rangle$**:** a user uid joins the group by engaging in an interactive protocol with the issuer $\mathcal{I}$. The user uid and issuer $\mathcal{I}$ perform algorithms CLS.Join and CLS.Issue respectively. These are input a state and an incoming message respectively, and output an updated state, an outgoing message, and a decision,

[1] For the sake of simplicity we state the algorithms for the setting where the requester and receiver of conversions is the same party, namely the verifier. However, our algorithms work in a public key setting to facilitate more general settings as well.

either cont, accept, or reject. The initial input to CLS.Join is the group public key, gpk, whereas the initial input to CLS.Issue is the issuer secret key isk, and the group public key gpk. If the user uid accepts, CLS.Join has a private output of $\mathbf{gsk}[uid]$.

CLS.Sign$(gpk, \mathbf{gsk}[uid], m) \to (\mu, \sigma)$**:** performed by the user with identifier uid, with input the group public key gpk, the user's secret key $\mathbf{gsk}[uid]$, and a message m; outputs a pseudonym μ and signature σ.

CLS.Verify$(gpk, m, \mu, \sigma) \to \{0, 1\}$**:** performed by the verifier $\mathcal{V}$; outputs 1 if σ is a valid signature on m for pseudonym μ under the group public key gpk, and 0 otherwise.

Blind Conversion. Finally, we want our pseudonymous group signatures to be blindly convertible. Thus, we introduce a dedicated CLS.Blind and CLS.Unblind procedure for the verifier and a CLS.Convert algorithm that requires the converter's secret key. The latter transforms the unlinkable pseudonyms in a consistent manner, i.e., outputting converted pseudonyms that are consistent whenever the input pseudonyms belong to the same user. The final pseudonyms after unblinding will be identical if and only if they stem from the same user and are obtained from the same CLS.Convert query. For example, if $(m_1, \mu_1), (m_2, \mu_2)$ stem from user A and $(m_3, \mu_3), (m_4, \mu_4), (m_5, \mu_5)$ stem from user B, then after blinding, conversion and unblinding a random shuffle of $(m_1, \overline{\mu_1}), (m_2, \overline{\mu_1}), (m_3, \overline{\mu_2})$, $(m_4, \overline{\mu_2}), (m_5, \overline{\mu_2})$ will be output.

CLS.Blind$(gpk, bpk, (\mu, m)) \to (c\mu, c)$**:** performed by the verifier $\mathcal{V}$, on input a pseudonym-message pair (μ, m), blinding public key bpk and group public key gpk; outputs a blinded pseudonym and message.

CLS.Convert$(gpk, csk, bpk, \{(c\mu_i, c_i)\}_k) \to \{(\overline{c\mu}_i, \overline{c}_i)\}_k$**:** performed by the converter $\mathcal{C}$, on input k blinded pseudonym-message tuples $\{(c\mu_i, c_i)\}_k = ((c\mu_1, c_1), ..., (c\mu_k, c_k))$, and the public blinding key bpk used; outputs converted pseudonyms $\{(\overline{c\mu}_i, \overline{c}_i)\}_k = ((\overline{c\mu}_1, \overline{c}_1), ..., (\overline{c\mu}_k, \overline{c}_k))$.

CLS.Unblind$(bsk, (\overline{c\mu}, \overline{c})) \to (\overline{\mu}, \overline{m})$**:** performed by the verifier $\mathcal{V}$, on input a converted pseudonym-message tuple and the blinding secret key bsk; outputs an unblinded converted pseudonym-message tuple $(\overline{\mu}, \overline{m})$.

We sometimes make the randomness r used in these algorithms explicit and, for example,

write $\mathsf{CLS.Blind}(gpk, bpk, (\mu, m); r)$.

4.2.2 Security Properties

We want CLS schemes to enjoy roughly the same security and privacy properties as group signatures when taking the added linkability into account. Defining these properties when pseudonyms can be *selectively* and *adaptively* converted is challenging, as it requires care to avoid trivial wins while keeping the adversary as powerful as possible.

In a nutshell, we require the following guarantees from convertibly linkable group signatures, where *(join) anonymity* and *non–transitivity* capture the privacy-related properties and *non–frameability* and *traceability* formalise the desired unforgeability.

(Join) Anonymity: Pseudonymous signatures should be unlinkable and untraceable (to a join session) even when the issuer and verifier are corrupt. When the converter is honest, unlinkability holds for all signatures for which the associated pseudonyms have not been explicitly linked through a conversion request. If the converter is corrupt and also controlled by the adversary, unlinkability is no longer possible, yet the anonymity of joins must remain.

Non–transitivity: Converted pseudonyms should be non-transitive, i.e., the verifier should not be able to link the outputs of different convert queries. Otherwise, a corrupt verifier would be able to gradually link together all pseudonyms that have ever been queried to the converter.

Conversion Blindness: The converter learns nothing about the pseudonyms (and messages) it receives and the transformed pseudonyms it computes.

Non–frameability: An adversary controlling the issuer and some corrupt users, should not be able to impersonate other honest users, i.e., create pseudonymous signatures that would be linked to a pseudonym of an honest user.

Traceability: An adversary should not be able to create more signatures that remain unlinkable in a conversion than they control corrupt users.

Clearly, any re-linked subset of the originally anonymous data increases the risk of re-

identification. Thus, the converter could enforce some form of access control to the re-linked data; e.g., only converting a certain amount of pseudonyms at once. The non-transitivity requirement then ensures that a corrupt verifier cannot further aggregate the individually learned data. We stress that these security properties only formalise the achievable privacy for the pseudonyms and signatures. They do not *and cannot* capture information leakage through the messages that the users sign. This is the case for all group signatures, and not special to our setting.

Oracles and State. The security notions we formalise in the following make use of a number of oracles which keep joint state, e.g., keeping track of queries and the set of corrupted parties. We present the detailed description of all oracles in Figure 4.1 and now provide an overview of them and their maintained records.

ADDU (join of honest user and honest issuer) Creates a new honest user for *uid* and internally runs a join protocol between the honest user and honest issuer. At the end, the honest user's secret key $\mathbf{gsk}[uid]$ is generated and from then on signing queries for *uid* will be allowed.

SNDU (join of honest user and corrupt issuer) Creates a new honest user for *uid* and runs the join protocol on behalf of *uid* with the corrupt issuer. If the join session completes, the oracle will store the user's secret key $\mathbf{gsk}[uid]$.

SNDI (join of corrupt user and honest issuer) Runs the join protocol on behalf of the honest issuer with corrupt users. For joins of honest users, the ADDU oracle must be used.

SIGN This oracle returns signatures for honest users that have successfully joined (via ADDU or SNDU, depending on the corruption setting).

CONVERT This oracle returns a set of converted pseudonyms along with their messages. To model that conversion is triggered by an at most honest-but-curious verifier, we request $\mathcal{V}$ to provide the unblinded set of pseudonyms along with signatures. The conversion will only be done when all signatures are valid. The oracle then internally blinds the pseudonym-message pairs and returns the blinded input, the randomness used for the blinding along with the converted output. When this oracle is used in the anonymity game, it further checks that the input does not allow the adversary

to trivially win by converting the challenge pseudonym together with pseudonyms from either of the challenge users.

All oracles have access to the following records maintained as global state:

HUL List of *uid*s of honest users, initially set to $\emptyset$. New honest users can be added by queries to the ADDU oracle (when the issuer is honest) or SNDU oracle (when the issuer is corrupt).

CUL List of corrupt users that have requested to join the group. Initially set to $\emptyset$, new corrupt users can be added through the SNDI oracle if the issuer is honest. If the issuer is corrupt, we do not keep track of corrupt users.

SL List of (uid, m, μ, σ) tuples requested from the SIGN oracle.

Helper Algorithms. We introduce two additional algorithms for notational simplicity in our security games: Identify and UnLink. Roughly, Identify allows one to test whether a pseudonym belongs to a certain *uid* by exploiting the convertibility of pseudonyms. That is, we create a second signature for $\mathbf{gsk}[uid]$ and use the converter's secret key to test whether both are linked. If so, Identify returns 1. This algorithm uses our second helper algorithm UnLink internally, which takes a list of pseudonym-message pairs and returns 1 if they are all unlinkable and 0 otherwise.

$\mathsf{Identify}(gpk, csk, uid, m, \mu)$

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{CLS.Sign}(gpk, \mathbf{gsk}[uid], 0)$

if $\mathsf{UnLink}(gpk, csk, ((\mu, m), (\mu', 0))) = 0$ **return** 1

else return 0

ADDU(uid)

if $uid \in \mathsf{HUL} \cup \mathsf{CUL}$ **return** $\perp$
$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp$
$\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, \mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow gpk$
$\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk)$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{CLS.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, \perp)$
while $\mathbf{dec}^{uid} = \mathsf{cont}$
  $(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Join}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{CLS.Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Issue}})$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{CLS.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Join}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return accept

SIGN(uid, m)

if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$
  return $\perp$
$(\mu, \sigma) \leftarrow_{\$} \mathsf{CLS.Sign}(gpk, \mathbf{gsk}[uid], m)$
$\mathsf{SL} \leftarrow \mathsf{SL} \cup \{(uid, m, \mu, \sigma)\}$
return (σ, μ)

SNDI(uid, M_{in})

if $uid \in \mathsf{HUL}$ **return** $\perp$
if $uid \notin \mathsf{CUL}$
  $\mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{uid\}$
  $\mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if undefined $\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk)$
$(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{CLS.Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{in}})$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

SNDU(uid, M_{in})

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}$
  $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if $\mathsf{st}^{uid}_{\mathsf{Join}}$ undefined $\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow gpk$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow_{\$} \mathsf{CLS.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

CONVERT($(\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k), bpk$)

if $\exists i \in [1, k]$ s.t. $\mathsf{CLS.Verify}(gpk, m_i, \mu_i, \sigma_i) = 0$ **return** $\perp$
if $bpk \notin \mathcal{BK}$ **return** $\perp$
if $\exists i$ s.t. $\mu_i = \mu^*$ and $\exists j \neq i$ s.t. $\mathsf{Identify}(uid^*_d, \mu_j) = 1$ for $d \in \{0, 1\}$
  return $\perp$
else compute $(c\mu_i, c_i) \leftarrow_{\$} \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i); r_i)$ for $i = 1, \ldots, k$
  and $\{(\overline{c\mu}_i, \overline{c}_i)\}_k \leftarrow_{\$} \mathsf{CLS.Convert}(gpk, csk, bpk, \{(c\mu_i, c_i)\}_k)$
return $(\{(c\mu_i, c_i)\}_k, \{(\overline{c\mu}_i, \overline{c}_i)\}_k, r_1, \ldots, r_k)$

Figure 4.1: Oracles used in our CLS model

$\mathsf{UnLink}(gpk, csk, ((\mu_1, m_1), ..., (\mu_k, m_k)))$

$(bpk, bsk) \leftarrow\!\!\$\ \mathsf{CLS.BKGen}(\mathsf{param})$

$\forall i \in [1,k] \quad (c\mu_i, c_i) \leftarrow\!\!\$\ \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i))$

$\{(\overline{c\mu}_i, \overline{c}_i)\}_k \leftarrow\!\!\$\ \mathsf{CLS.Convert}(gpk, csk, bpk, \{(c\mu_i, c_i)\}_k)$

$\forall i \in [1,k] \quad (\overline{\mu}_i, \overline{m}_i) \leftarrow \mathsf{CLS.Unblind}(bsk, (\overline{c\mu}_i, \overline{c}_i))$

if $\exists (i,j)$ with $i \neq j$ s.t. $\overline{\mu}_i = \overline{\mu}_j$ **return** 0

else return 1

For even more simplicity we often omit the keys for the algorithms (as they are clear from the context). That is, we write $\mathsf{Identify}(uid, \mu)$ which will indicate whether the pseudonym μ belongs to the user with identity uid or not. Likewise, we write $\mathsf{UnLink}(\mu_1, \ldots, \mu_k)$ to test whether all pseudonyms are uncorrelated or not.

Correctness. CLS signatures, generated by honest parties, should be correct and consistent. More precisely, we formulate correctness via three requirements. *Correctness of sign* guarantees that signatures formed using the CLS.Sign algorithm with a user secret key generated honestly will verify correctly. *Correctness of conversion* guarantees that after blinding, converting and then unblinding correctly, the output will be correctly linked messages/ pseudonyms. The messages output after unblinding should be the original messages shuffled with a permutation Π (the permutation is necessary for non–transitivity to hold). The unblinded converted pseudonyms should be shuffled with the same permutation and are identical if and only if they stem from the same user. *Consistency* is a stronger variant of conversion-correctness and requires that the correlations of pseudonyms established through the conversion procedure must be consistent across queries. More precisely, if a conversion query reveals that two pseudonym μ_1 and μ_2 are linked, and another one that μ_2 and μ_3 are linked, then it must also hold that a conversion query for μ_1 and μ_3 returns linked pseudonyms. We require that this property even holds for maliciously formed pseudonyms, which will be a helpful property in some of our security proofs.

The detailed definitions for correctness are given in Figure 4.2 and follow the game-based style already used in [14] for correctness definitions.

Definition 4.2 (Correctness)**.** A CLS scheme satisfies correctness if, for all adversaries $\mathcal{A}$, $\Pr[\mathbf{Exp}^{corr-sig}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1] = 0$, $\Pr[\mathbf{Exp}^{corr-conv}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1] \leq \mathsf{negl}(\tau)$, and, $\Pr[\mathbf{Exp}^{consist}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1] = 0$.

For the *correctness of conversion*, the negligible chance that the adversary has of winning corresponds to the negligible chance that multiple user identifiers have the same secret key.

Anonymity *(Corrupt Issuer and Verifier).* This security requirement captures the desired anonymity properties when the converter is honest and the issuer is corrupt. The verifier is honest–but–curious, which we model by only allowing unblinded valid signatures to be input to the conversion oracle, which can be verified. Just as in conventional group signatures, we want that the signatures of honest users are unlinkable and cannot be traced back to a user's join session with the corrupt issuer. To model this property, we let the adversary output uids of two honest users together with a message and return a challenge (μ^*, σ^*) that is created either by user uid_0 or uid_1. For anonymity, the adversary should not be able to determine the user's identity better than by guessing.

In our setting, this property must hold when the adversary has access to the conversion oracle where it can obtain linked subsets of the pseudonymous data. To avoid trivial wins, the adversary is not allowed to make conversion queries that link the challenge pseudonym μ^* to another pseudonym belonging to one of the two honest challenge users.

Definition 4.3 (Anonymity)**.** A CLS scheme satisfies anonymity if for all polynomial–time adversaries $\mathcal{A}$ the following advantage is negligible in τ :

$$\left|\Pr[\mathbf{Exp}^{anon-0}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1] - \Pr[\mathbf{Exp}^{anon-1}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1]\right|.$$

Experiment: $\mathbf{Exp}^{anon-b}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \emptyset$

$(uid_0^*, uid_1^*, m^*, \mathsf{st}) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU,SIGN,CONVERT}}(\mathsf{choose}, gpk, isk)$

if $uid_0^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0^*] = \perp$ or $uid_1^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_1^*] = \perp$ **return** 0

$(\mu^*, \sigma^*) \leftarrow_\$ \mathsf{CLS.Sign}(gpk, \mathbf{gsk}[uid_b^*], m^*)$

$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU,SIGN,CONVERT}}(\mathsf{guess}, \mathsf{st}, \mu^*, \sigma^*)$

return b^*

Non–transitivity *(Corrupt Issuer and Verifier).* The second privacy-related property we want to guarantee is the strict non-transitivity of conversions. This ensures that

Experiment: $\mathbf{Exp}^{corr-sig}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL} \leftarrow \emptyset, (uid, m) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU}}(gpk)$
if $\mathbf{gsk}[uid] = \perp$ **return** 0
$(\mu, \sigma) \leftarrow_\$ \mathsf{CLS.Sign}(gpk, \mathbf{gsk}[uid], m)$
if $\mathsf{CLS.Verify}(gpk, m, \mu, \sigma) = 0$ **return** 1 **else return** 0

Experiment: $\mathbf{Exp}^{corr-conv}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL} \leftarrow \emptyset, (bpk, bsk) \leftarrow_\$ \mathsf{CLS.BKGen}(\mathsf{param})$
$((uid_1, m_0), ..., (uid_k, m_k)) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU}}(gpk)$
if $\exists i \in [1, k]$ st $\mathbf{gsk}[uid_i] = \perp$ **return** 0
$\forall i \in [1, k] \quad (\mu_i, \sigma_i) \leftarrow_\$ \mathsf{CLS.Sign}(gpk, \mathbf{gsk}[uid_i], m_i)$
$\forall j \in [1, k] \quad (c\mu_j, c_j) \leftarrow_\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_j, m_j))$
$\{(\overline{c\mu}_i, \overline{c}_i)\}_k \leftarrow_\$ \mathsf{CLS.Convert}(gpk, csk, bpk, \{(c\mu_i, c_i)\}_k; \Pi)$
$\forall j \in [1, k] \quad (\overline{\mu}_j, \overline{m}_j) \leftarrow \mathsf{CLS.Unblind}((\overline{c\mu}_j, \overline{c}_j), bsk)$
if $\exists$ permutation $\Pi : [1, k] \rightarrow [1, k]$ s.t.
1.$\forall i \in [1, k] \quad \overline{m}_{\Pi(i)} = m_i$
2.$\forall (i, j) \in [1, k]$ with $uid_i = uid_j \quad \overline{\mu}_{\Pi(i)} = \overline{\mu}_{\Pi(j)}$
3.$\forall (i, j) \in [1, k]$ with $uid_i \neq uid_j \quad \overline{\mu}_{\Pi(i)} \neq \overline{\mu}_{\Pi(j)}$
return 0
else return 1

Experiment: $\mathbf{Exp}^{consist}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$((m_0, \mu_0, \sigma_0), (m_1, \mu_1, \sigma_1), (m_2, \mu_2, \sigma_2)) \leftarrow_\$ \mathcal{A}(gpk, isk, csk)$
if $\mathsf{UnLink}(\mu_0, \mu_1) = 1$ or $\mathsf{UnLink}(\mu_1, \mu_2) = 1$ **return** 0
if $\mathsf{UnLink}(\mu_0, \mu_2) = 1$ **return** 1
else return 0

Figure 4.2: Security games for correctness of CLS

the outputs of separate convert queries cannot be linked together, further than what is already possible due the messages queried. For example, if $\overline{\mu}_1$ and $\overline{\mu}_2$ are outputs from two separate convert queries, the adversary should not be able to decide whether they were derived from the same pseudonym or not. Otherwise the verifier could gradually build lists of linked pseudonyms, adding to these during every convert query and eventually recover the linkability among all pseudonymous signatures. This requirement should hold if the issuer is corrupt, the converter is honest, and the verifier is honest–but–curious, therefore again we only allow unblinded valid signatures to be input to the conversion oracle.

To model non-transitivity of conversions we use a simulation-based approach, requiring the indistinguishability of an ideal and a real world. In the real world, all convert queries are handled normally through the CONVERT oracle defined in Figure 4.1. Whereas in the ideal world, the converted pseudonyms are produced by a simulator SIM through the CONVSIM oracle. For a conversion request of input $(\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k)$, the simulator will only learn which of the messages belong together, i.e., are associated to pseudonyms that belong to the same user *uid*. For *honest* users, this can be looked up through the records of the signing oracle that stores tuples $(uid, m_i, \mu_i, \sigma_i)$ for each signing query. Thus, we let the simulator mimic the conversion output for all pseudonyms stemming from honest users, and convert pseudonyms from corrupt users normally (as there is no privacy to guarantee for them anyway). Finally, the CONVSIM oracle outputs a random shuffle of the correctly converted pseudonyms of corrupt users, and the simulated ones for honest users. As mentioned before, we assume the verifier to be honest-but-curious, which we enforce by requesting the adversary to output valid signatures along with the pseudonyms to be converted and handle the blinding step within the conversion oracle.

Definition 4.4 (Non–transitivity)**.** A CLS scheme satisfies non-transitivity if for all polynomial–time adversaries $\mathcal{A}$ there exists an efficient simulator SIM such that the following advantage is negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{nontrans-0}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1] - \Pr[\mathbf{Exp}^{nontrans-1}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1]\right|.$$

Experiment: $\mathbf{Exp}^{nontrans-b}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow\!\!\$\ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow\!\!\$\ \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \emptyset$

$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU,SIGN,CONVX}}(\mathsf{guess}, gpk, isk)$

where the oracle CONVX works as follows:

if $b = 0$ (real world) **then** CONVX is the standard CONVERT oracle

if $b = 1$ (ideal world) **then** CONVX is the simulated CONVSIM oracle

return b^*

$\mathsf{CONVSIM}((\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k), bpk)$

if $\exists i \in [1,k]$ s.t. $\mathsf{CLS.Verify}(gpk, m_i, \mu_i, \sigma_i) = 0$ or $bpk \notin \mathcal{BK}$ **return** $\perp$

Set $\mathsf{CL} \leftarrow \emptyset$

Compute $(c\mu_i, c_i) \leftarrow\!\!\$\ \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i); r_i)$ for $i = 1, \ldots, k$

$\forall i \in [1,k]$ // determine message clusters L_{uid} for honest users and list CL of corrupt pseudonyms

if $(uid, m_i, \mu_i, \sigma_i) \in \mathsf{SL}$ // pseudonyms from honest users

if L_{uid} does not exist, create $\mathsf{L}_{uid} \leftarrow \{m_i\}$ **else** set $\mathsf{L}_{uid} \leftarrow \mathsf{L}_{uid} \cup \{m_i\}$

else $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(c_i, c\mu_i)\}$ // pseudonyms from corrupt users

$\{(\overline{c\mu}_i, \overline{c}_i)\}_{i=1,\ldots k'} \leftarrow\!\!\$\ \mathsf{CLS.Convert}(gpk, csk, bpk, \mathsf{CL})$ for $k' \leftarrow |\mathsf{CL}|$ // normally convert corrupt μs

Let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters

$\{(\overline{c\mu}_i, \overline{c}_i)\}_{i=k'+1,\ldots k} \leftarrow\!\!\$\ \mathsf{SIM}(gpk, bpk, \mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}})$ // simulate conversion for honest μs

Let $\{(\overline{c\mu}'_i, \overline{c}'_i)\}_{i=1,\ldots k}$ be a random permutation of $\{(\overline{c\mu}_i, \overline{c}_i)\}_{i=1,\ldots k}$

return $(\{(c\mu_i, c_i, r_i)\}_{i=1,\ldots,k}, \{(\overline{c\mu}'_i, \overline{c}'_i)\}_{i=1,\ldots k}, r_1, \ldots, r_k)$

Anonymity versus Non–transitivity. Note that non-transitivity is not covered by the anonymity notion defined before. A scheme that satisfies anonymity could output the converted pseudonyms in the exact same order as the input ones, allowing trivial linkage between the in- and output of each conversion request. Thus, whenever the same pseudonym is used as input to several conversion queries, this would enable the linkability of the transformed pseudonyms across the different conversions, which is exactly what non-transitivity aims to avoid. On the first glance, it might seem odd that having transitive conversions does not harm our anonymity property. However, transitivity is

only useful when *several* pseudonyms belonging to the same user appear in each conversion request with one pseudonym being re-used in all these sessions. In the anonymity game, the challenge pseudonym is not allowed to be used in combination with any other pseudonym stemming from either of the challenge users (as this would make the definition unachievable), and thus the transitivity of conversions can not be exploited.

Conversion Blindness *(Corrupt Issuer and Converter).* A crucial property of our signatures is that they can be converted in an oblivious manner, i.e., without the converter learning anything about the pseudonyms or messages it converts. In particular, this blindness property ensures the unlinkability of blinded inputs across several conversion requests. Conversion blindness should hold if both the issuer and converter are corrupt, but the verifier is honest. We formalise this property in a classic indistinguishability style: the adversary outputs two tuples of pseudonym-message pairs and receives a blinded version of either of them. Given that blinding of pseudonyms is a public-key operation, we do not need an additional blinding oracle. In fact, we do not provide the adversary with *any* oracle access at all in this game. They already have corrupted the issuer and converter, and this property does not distinguish between honest and corrupt users, thus we simply assume that the adversary also has full control over all users.

Definition 4.5 (Conversion Blindness)**.** A CLS scheme satisfies conversion blindness if for all polynomial–time adversaries $\mathcal{A}$ the following advantage is negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{blind-conv-0}_{\mathcal{A},\mathsf{CLS}}(\tau)=1]-\Pr[\mathbf{Exp}^{blind-conv-1}_{\mathcal{A},\mathsf{CLS}}(\tau)=1]\right|.$$

Experiment: $\mathbf{Exp}^{blind-conv-b}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk),\ (bpk, bsk) \leftarrow_\$ \mathsf{CLS.BKGen}(\mathsf{param})$

$(\mathsf{st}, (\mu_0, m_0), (\mu_1, m_1)) \leftarrow_\$ \mathcal{A}(\mathsf{choose}, gpk, isk, csk, bpk)$

$(c\mu^*, c^*) \leftarrow_\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_b, m_b))$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, c\mu^*, c^*)$

return b^*

Join Anonymity *(Corrupt Issuer, Converter and Verifier).* The final privacy related property we require from a CLS scheme is the anonymity of joins even if *all* central entities are corrupt. Here the challenge is that the adversary, controlling the issuer,

converter and verifier, should not be able to link signatures of an honest user back to a particular join session. This is the best one can hope for in this corruption setting as unlinkability of signatures (as guaranteed by our anonymity property) is no longer possible: the corrupt converter can simply convert all signatures/pseudonyms into a consistent representation. Such a property does not exist in conventional group signatures, as a corrupt opener can always reveal the join identity. In our setting, signatures can only be linked instead of being opened and thus anonymity of the join procedure can be preserved.

To model this property we let the adversary output two identities of honest users uid_0, uid_1 that have successfully joined. We then provide the adversary access to a signing oracle for one of them. This is done by adding the challenge user uid^* (where uid^* stands for a dummy handle) to the list of honest users HUL with user secret key $\mathbf{gsk}[uid_b]$. Thus, in the second stage of the game, the adversary can invoke the SIGN oracle on uid^* to receive signatures of messages of their choice for the challenge user. The adversary wins if they can determine the bit b better than by guessing. To avoid trivial wins, the adversary is not allowed to see any signature directly from uid_0 or uid_1.

Definition 4.6 (Join Anonymity)**.** A CLS scheme satisfies join anonymity if for all polynomial–time adversaries $\mathcal{A}$ the following advantage is negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{anon-join-0}_{\mathcal{A},\mathsf{CLS}}(\tau)=1]-\Pr[\mathbf{Exp}^{anon-join-1}_{\mathcal{A},\mathsf{CLS}}(\tau)=1]\right|.$$

Experiment: $\mathbf{Exp}^{anon-join-b}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \emptyset$

$(\mathsf{st}, uid_0, uid_1) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(\mathsf{choose}, gpk, isk, csk)$

if uid_0 or $uid_1 \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0] = \perp$ or $\mathbf{gsk}[uid_1] = \perp$ **return** 0

Choose $uid^*, \mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid^*\}, \mathbf{gsk}[uid^*] \leftarrow \mathbf{gsk}[uid_b]$

$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(\mathsf{guess}, \mathsf{st}, uid^*)$

return b^* **if** $(uid_d, *) \notin \mathsf{SL}$ for $d = 0, 1$ **else return** 0

Non–frameability ***(Corrupt Issuer, Converter and User)*** This notion captures the desired unforgeability properties when the issuer, converter, verifier and some of the users are corrupt, and requires that an adversary should not be able to impersonate an honest user. Our definition is very similar to the non-frameability definitions in standard group

signature or DAA schemes [12, 14, 15]. Roughly, the only part we have to change is how we detect that an honest user has been framed. In group signatures, non-frameability exploits the presence of the group manager that can open signatures and requires that an adversary cannot produce signatures that will open to an honest user who has not created this signature. Here we have the converter instead of the group manager (or dedicated opening authority), and thus express non-frameability through the linkage that is created in a conversion. More precisely, an adversary should not be able to produce a valid signature (μ^*, σ^*) that within a conversion request would falsely link to a signature of an honest user. For generality (and sake of brevity), we use our helper function $\mathsf{Identify}$ that we introduced at the beginning of this section to express that the adversary's signature should not be recognised as a signature of an honest user. For our unforgeability guarantees we assume an honest but curious converter. This materialises in our requirement by using the $\mathsf{Identify}$ function, which honestly performs $\mathsf{CLS.Convert}$, to determine whether a forgery has occurred.

Definition 4.7 (Non–frameability)**.** A CLS scheme satisfies non–frameability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{nonframe}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathbf{Exp}^{nonframe}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\, \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow\!\!\$\, \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow\!\!\$\, \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \emptyset$

$(uid, m^*, \mu^*, \sigma^*) \leftarrow\!\!\$\, \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(gpk, isk, csk)$

return 1 **if** all of the following conditions are satisfied:

$\mathsf{CLS.Verify}(gpk, m^*, \mu^*, \sigma^*) = 1$ and

$\mathsf{Identify}(uid, m^*, \mu^*) = 1$ where $uid \in \mathsf{HUL}$ and

$(uid, m^*, \mu^*, \sigma^*) \notin \mathsf{SL}$

Traceability ***(Corrupt Converter and User)*** Our final requirement formalises the unforgeability properties when only the converter, verifier and some users are corrupt, whereas the issuer and some users are honest. In this setting, an adversary should not be able to output more pseudonymous signatures that remain unlinkable in a conversion than the number of users it has corrupted. This is again an adaptation of the existing traceability notions for group signatures with an opening authority [12, 14] or user-controlled linkability [15]. Interestingly, in the latter work given in Section 2.8.1 (that is closer to our setting than standard group signatures), two traceability notions were introduced. While

one is similar in spirit to our notion, a second property guarantees that all signatures of corrupt users can be traced back to the exact signing key that the corrupt user has established in the join protocol with the honest issuer. This seems to be an odd requirement, as it is not noticeable in the real world. In fact, we do not limit the strategy of the adversary in that way and only require their power to be bounded by the amount of corrupt users they control.

Our definition uses our helper algorithm UnLink that we introduced at the beginning of this section and that internally uses the $\mathsf{CLS.Convert}$ algorithm to detect whether pseudonyms are unlinkable or not. Note that UnLink returns 1 only if *all* inputs are mutually unlinkable, i.e., there is not a single tuple of input pseudonyms that was converted to the same value. As in non–frameability we assume an honest but curious converter. This materialises in this requirement by using the UnLink function, which honestly performs $\mathsf{CLS.Convert}$, to determine whether signatures are unlinkable.

Definition 4.8 (Traceability)**.** A CLS scheme satisfies traceability if for all polynomial–time adversaries $\mathcal{A}$ the advantage $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CLS}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow\!\!\$\ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow\!\!\$\ \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \emptyset$

$((m_1, \mu_1, \sigma_1), ..., (m_k, \mu_k, \sigma_k)) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{ADDU,SNDI,SIGN}}(gpk, csk)$

return 1 if all of the following conditions are satisfied:

$\forall j \in [1, k] : \mathsf{CLS.Verify}(gpk, m_j, \mu_j, \sigma_j) = 1$ and $(*, m_j, \mu_j, \sigma_j) \notin \mathsf{SL}$ and

$k > |\mathsf{CUL}|$ and

$\mathsf{UnLink}(gpk, csk, ((\mu_1, m_1), ..., (\mu_k, m_k))) = 1$

4.3 Our CLS Construction

We now present our construction that securely realises such CLS group signatures. Our scheme follows the classical *sign-and-encrypt* paradigm: we use BBS+ signatures [9], as discussed in Section 2.5.6, for attesting group membership, i.e., a user will blindly receive a BBS+ signature from the group issuer on the user's secret key y. To sign a message m on behalf of the group, the user computes a SPK for m, where they prove knowledge

of a BBS+ signature on y, and also encrypts h^y under the converter's public key. The ElGamal ciphertext that encrypts h^y serves as the associated pseudonym μ.

We discuss the homomorphic and re-randomisable properties of ElGamal encryption in Section 2.5.2. To blind a set of k pseudonym-message pairs $(\mu_1, m_1), \ldots (\mu_k, m_k)$ for conversion, the verifier encrypts each value under its own ElGamal public key. As the pseudonyms are already ElGamal ciphertexts themselves, this results in a nested double-encryption of h^y being encrypted under both keys. The converter then decrypts the "inner" part of the ciphertext and blindly raises the result to a random value r. This r is chosen fresh for every conversion query, but used consistently within. That is, all pseudonyms belonging to the same user will be mapped to the same query-specific DDH tuple h^{yr}. Finally, the converter re-randomises all ciphertexts and shuffles them to destroy any linkage between the in- and output tuples — this is crucial for achieving the desired non-transitivity property. The verifier then simply decrypts the received tuples and can link correlated data via the converted pseudonyms $\overline{c\mu}_i$.

4.3.1 Detailed Description of CLS–DDH

Setup and Key Generation. We use a type–3 bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ with g_1 and g_2 being generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Further, we need four additional generators g, h and h_1, h_2 in $\mathbb{G}_1$, where h_1, h_2 are used for the BBS+ signature and g, h will be used for the ElGamal encryption.

$\mathsf{CLS.Setup}(1^\tau)$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$\ \mathcal{G}(1^\tau), g, h, h_1, h_2 \leftarrow\!\!\$\ \mathbb{G}_1$

return $\mathsf{param} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, g_2, g, h, h_1, h_2)$

$\mathsf{CLS.IKGen}(\mathsf{param})$

$isk \leftarrow\!\!\$\ \mathbb{Z}_p^*, ipk \leftarrow g_2^{isk}$

return (ipk, isk)

$\mathsf{CLS.CKGen}(\mathsf{param})$

$csk \leftarrow\!\!\$\ \mathbb{Z}_p^*, cpk \leftarrow g^{csk}$

return (cpk, csk)

$\mathsf{CLS.BKGen}(\mathsf{param})$

$bsk \leftarrow\!\!\$\ \mathbb{Z}_p^*, bpk \leftarrow g^{bsk}$

return (bpk, bsk)

Join. To join the group, users perform an interactive protocol with the issuer to obtain their user secret key and group credential. Roughly, the $\mathbf{gsk}[uid]$ of a user consists of a secret key $y \in \mathbb{Z}_p^*$ and a BBS+ signature (A, x, s) of $\mathcal{I}$ on y, where $A = (g_1 h_1^y h_2^s)^{1/(isk+x)}$. The detailed protocol of $\langle \mathsf{CLS.Join}(gpk), \mathsf{CLS.Issue}(isk, gpk) \rangle$ is given in Figure 4.3.

$\mathcal{U}.\mathsf{CLS.Join}(gpk)$	$\rightleftharpoons$	$\mathcal{I}.\mathsf{CLS.Issue}(isk, gpk)$
		choose $n \leftarrow_\$ \{0,1\}^\tau$
	$\xleftarrow{n}$	
$y \leftarrow_\$ \mathbb{Z}_p^*, H \leftarrow h_1^y,$		
$\pi_H \leftarrow_\$ \mathsf{SPK}\{(\underline{y}) : H = h_1^y\}(n)$		
	$\xrightarrow{H, \pi_H}$	
		Verify π_H and that $H \in \mathbb{G}_1$
		$x \leftarrow_\$ \mathbb{Z}_p^*, s \leftarrow_\$ \mathbb{Z}_p^*$
		$A \leftarrow (H h_2^s g_1)^{1/(isk+x)}$
	$\xleftarrow{A, x, s}$	
check that $A \neq 1_{\mathbb{G}_1}$ and that		
$e(A, g_2)^x e(A, ipk) = e(g_1 H h_2^s, g_2)$		
return $\mathbf{gsk}[uid] \leftarrow (A, x, y, s)$		

Figure 4.3: Join protocol of our CLS–DDH construction

Sign and Verify. To sign a message m under $\mathbf{gsk}[uid] = (A, x, y, s)$, the user proves knowledge of a BBS+ credential (A, x, s) on its secret key y and also encrypts h^y under the converter's public key cpk. The proof π proves knowledge of the BBS+ credential and asserts that the encryption contains the same value y. From π we only need the value y to be online extractable. We use the improved proof of knowledge of a BBS+ credential from Camenisch et al. [32], who have shown how to move most of the work from $\mathbb{G}_T$ to $\mathbb{G}_1$ and thus yield a much faster instantiation than the original proof by Au et al. [9]. For verification, one verifies the proof π and some correctness properties of the re-randomised versions of A that are sent along with the proof. In more detail, CLS.Sign and CLS.Verify are defined as follows:

$\mathsf{CLS.Sign}(gpk, \mathbf{gsk}[uid], m)$

parse $\mathbf{gsk}[uid] = (A, x, y, s)$, $gpk = (ipk, cpk)$

$\alpha \leftarrow\!\!\$\, \mathbb{Z}_p^*$, $\mu_1 \leftarrow g^\alpha$, $\mu_2 \leftarrow cpk^\alpha h^y$, $r_1, r_2 \leftarrow\!\!\$\, \mathbb{Z}_p^*$,

$A' \leftarrow A^{r_1}, \hat{A} \leftarrow A'^{-x}(g_1 h_1^y h_2^s)^{r_1}, d \leftarrow (g_1 h_1^y h_2^s)^{r_1} h_2^{-r_2}, r_3 \leftarrow r_1^{-1}, s' \leftarrow s - r_2 r_3$

$\pi \leftarrow\!\!\$\, \mathsf{SPK}\{(x, \underline{y}, r_2, r_3, s', \alpha) : \mu_1 = g^\alpha \wedge \mu_2 = cpk^\alpha h^y \wedge \hat{A}/d = A'^{-x} h_2^{r_2} \wedge g_1 h_1^y = d^{r_3} h_2^{-s'}\}(m)$

$\sigma \leftarrow (A', \hat{A}, d, \pi), \mu \leftarrow (\mu_1, \mu_2)$

return (μ, σ)

$\mathsf{CLS.Verify}(gpk, m, \mu, \sigma)$

parse $\sigma = (A', \hat{A}, d, \pi)$

return 1 **if** $A' \neq 1_{\mathbb{G}_1}, e(A', ipk) = e(\hat{A}, g_2)$,

and π holds for $A', \hat{A}, d, \mu, m$ w.r.t. gpk

Blind Conversions. When the verifier wants to learn which of the pseudonymously received messages belong together, it sends a batch of pseudonym-message pairs in blinded form to the converter. That is, it encrypts the messages and pseudonyms using ElGamal encryption. The pseudonyms are ElGamal ciphertexts already and we roughly wrap them in another encryption layer. The converter then blindly decrypts the pseudonyms, i.e., decrypts the "inner" part of the ciphertext, which yields h^y encrypted under the verifiers blinding key bpk. To ensure non-transitivity, i.e., restrict the linkage of pseudonyms to hold only within the queried batch, the converter blindly raises the encrypted h^y to a random exponent r. This value is chosen afresh for every batch but used consistently within the query, i.e., all pseudonyms that belong to the same user with secret key y will be mapped consistently to h^{yr}. To ensure that the ciphertexts and their order cannot leak any additional information, we let the converter re-randomise and shuffle all ciphertexts before they returns them to the verifier. Both the verifier and the converter are assumed to be at most honest-but-curious, and so proofs that they have performed CLS.Blind and CLS.Convert correctly are not needed.

CLS.Blind(gpk, bpk, μ, m)

parse $gpk = (ipk, cpk),\ \mu = (\mu_1, \mu_2)$

$\alpha, \beta, \gamma \leftarrow_\$ \mathbb{Z}_p^*, c\mu_1 \leftarrow \mu_1 g^\beta,\ c\mu_2 \leftarrow g^\alpha,\ c\mu_3 \leftarrow \mu_2 cpk^\beta bpk^\alpha$

$c_1 \leftarrow g^\gamma, c_2 \leftarrow bpk^\gamma m, c\mu \leftarrow (c\mu_1, c\mu_2, c\mu_3),\ c \leftarrow (c_1, c_2)$

return $(c\mu, c)$

CLS.Convert$(gpk, csk, bpk, ((c\mu_1, c_1), ..., (c\mu_k, c_k)))$

parse $c\mu_i = (c\mu_{i,1}, c\mu_{i,2}, c\mu_{i,3}),\ c_i \leftarrow (c_{i,1}, c_{i,2}),\ r \leftarrow_\$ \mathbb{Z}_p^*$

for $i = 1, \ldots k$:

$c\mu'_{i,1} \leftarrow c\mu_{i,2}^r,\ c\mu'_{i,2} \leftarrow (c\mu_{i,3} c\mu_{i,1}^{-csk})^r$ // decrypt μ and raise to r

$r_1, r_2 \leftarrow_\$ \mathbb{Z}_p^*$ // re-randomise all ciphertexts

$c\mu''_{i,1} \leftarrow c\mu'_{i,1} g^{r_1},\ c\mu''_{i,2} \leftarrow c\mu'_{i,2} bpk^{r_1}$

$c'_{i,1} \leftarrow c_{i,1} g^{r_2},\ c'_{i,2} \leftarrow c_{i,2} bpk^{r_2}$

choose random permutation Π, **for** $i = 1, \ldots, k$: $(\overline{c\mu}_i, \overline{c}_i) \leftarrow (c\mu''_{\Pi(i)}, c'_{\Pi(i)})$

return $((\overline{c\mu}_1, \overline{c}_1), ..., (\overline{c\mu}_k, \overline{c}_k))$

CLS.Unblind$(bsk, (\overline{c\mu}, \overline{c}))$

parse $\overline{c\mu} = (\overline{c\mu}_1, \overline{c\mu}_2),\ \overline{c} = (\overline{c}_1, \overline{c}_2)$

$\overline{\mu} \leftarrow \overline{c\mu}_2 \overline{c\mu}_1^{-bsk},\ m \leftarrow \overline{c}_2 \overline{c}_1^{-bsk}$

return $(\overline{\mu}, m)$

4.4 Security of CLS–DDH

We now show that our scheme satisfies all security properties defined in Section 4.2. More precisely, we show that the following theorem holds (using the type–3 pairing version of the q-SDH assumption given in [20]).

Theorem 4.1. The CLS–DDH construction presented in Section 4.3.1 is a secure CLS as defined in Section 4.2 if the DDH assumption holds in $\mathbb{G}_1$, the q-SDH assumption holds, and the SPK is simulation-sound, zero-knowledge and online extractable (for the underlined values).

4.4.1 Correctness

The construction satisfies **correctness of sign** due to the correctness of the signature zero–knowledge proofs used, and because $\hat{A} = A^{-xr_1}A^{r_1(isk+x)} = A^{r_1 isk} = A'^{isk}$.

Let (A_i, x_i, y_i, s_i) be the secret signing key for the user with identifier uid_i. Then $\forall i \in k$, for a_i, a_i', a_i'' chosen randomly, $\mu_i = (g^{a_i}, cpk^{a_i}h^{y_i})$, $c\mu_i = (g^{a_i+a_i'}, g^{a_i''}, cpk^{a_i+a_i'}h^{y_i}bpk^{a_i''})$. Then for $r, r_i' \leftarrow_\$ \{0,1\}^*$ and a random permutation Π, $\overline{c\mu_{\Pi(i)}} = (g^{a_i''r+r_i'}, h^{y_i r}bpk^{a_i''r+r_i'})$, and $\overline{\mu_{\Pi(i)}} = h^{y_i r}$. Therefore $\overline{\mu_{\Pi(i)}} = \overline{\mu_{\Pi(j)}}$ if and only if $y_i = y_j$. Except with negligible probability, as y is chosen randomly, this only occurs if and only if $uid_i = uid_j$. $\forall i \in [1,k]$, for α chosen randomly, $c_i = (g^\alpha, m_i bpk^\alpha)$, for r_i'' chosen randomly, $\bar{c}_{\Pi(i)} = (g^{\alpha+r_i''}, m_i bpk^{\alpha+r_i''})$, therefore $\overline{m}_{\Pi(i)} = m_i$. Therefore, the construction satisfies **correctness of conversion**.

Assume $\mathsf{UnLink}(gpk, csk, ((\mu_0, m_0), (\mu_1, m_1))) = 0$ and $\mathsf{UnLink}(gpk, csk, ((\mu_1, m_1), (\mu_2, m_2))) = 0$. Due to the same argument, letting r_1, r_2 be the randomness chosen in $\mathsf{CLS.Convert}$, $(\mu_{0,2}\mu_{0,1}^{-csk})^{r_1} = (\mu_{1,2}\mu_{1,1}^{-csk})^{r_1}$ and $(\mu_{1,2}\mu_{1,1}^{-csk})^{r_2} = (\mu_{2,2}\mu_{2,1}^{-csk})^{r_2}$. Therefore, $\mu_{0,2}\mu_{0,1}^{-csk} = \mu_{2,2}\mu_{2,1}^{-csk}$. However, if $\mathsf{UnLink}(gpk, csk, ((\mu_0, m_0), (\mu_2, m_2))) = 1$, then $(\mu_{0,2}\mu_{0,1}^{-csk})^{r_3} \neq (\mu_{2,2}\mu_{2,1}^{-csk})^{r_3}$, where r_3 was chosen during $\mathsf{Convert}$. This is a contradiction. Therefore, the construction satisfies **consistency**.

4.4.2 Anonymity

Lemma 4.1. The CLS–DDH construction presented in Section 4.3.1 satisfies **anonymity** if the DDH assumption holds in $\mathbb{G}_1$, and the SPK is unbounded simulation-sound, zero–knowledge and online extractable (for the underlined values).

Proof. We assume that an adversary $\mathcal{A}$ exists that makes q queries to the SNDU oracle for distinct user identifiers and q_{conv} queries to the CLS.Convert oracle, guesses b correctly in the anonymity game and wins with probability $\epsilon + 1/2$.

We define Game (0,0) to be the anonymity experiment, with b chosen randomly at the beginning, using the CLS–DDH construction. Let $P_{0,0}$ be the event that an adversary $\mathcal{A}$ correctly guesses b after Game (0,0).

$\mathsf{CONVERT}((\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k), bpk)$

if $\exists i \in [1,k]$ s.t. $\mathsf{CLS.Verify}(gpk, m_i, \mu_i, \sigma_i) = 0$ **return** $\perp$
if $bpk \notin \mathcal{BPK}$ **return** $\perp$
if $\exists i$ s.t. $\mu_i = \mu^*$ and $\exists j \neq i$ s.t. $\mathsf{Identify}(uid_d^*, \mu_j) = 1$ for $d \in \{0,1\}$
return $\perp$
else compute $(c\mu_i, c_i) \leftarrow_\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i); r_i')$ for $i = 1, \ldots, k$
$r \leftarrow_\$ \mathbb{Z}_p^*, \forall i \in [1,k]$
$\alpha \leftarrow_\$ \mathbb{Z}_p^*, c_i' \leftarrow (c_{i,1} g^\alpha, c_{i,2} bpk^\alpha)$
if $\mu_i = \mu^*$ $\mu_i' \leftarrow_\$ \mathbb{G}_1, \beta \leftarrow_\$ \mathbb{Z}_p^*, c\mu_i' \leftarrow (g^\beta, \mu_i' bpk^\beta)$
if $\mu_i \neq \mu^*$
$\beta \leftarrow_\$ \mathbb{Z}_p^*, c\mu_{i,1}' \leftarrow c\mu_{i,2}^r g^\beta, c\mu_{i,2}' \leftarrow c\mu_{i,3}^r c\mu_{i,1}^{-rcsk} bpk^\beta$
Choose random permutation $\Pi; \forall i \in [1,k]$ $(\overline{c\mu}_i, \overline{c}_i) \leftarrow (c\mu_{\Pi(i)}', c_{\Pi(i)}')$
return $(\{(c\mu_i, c_i)\}_k, \{(\overline{c\mu}_i, \overline{c}_i)\}_k, r_1', \ldots, r_k')$

Figure 4.4: Convert oracle used during the first j queries of Game $(0,j)$ in the CLS anonymity proof

Game $(0,j)$ is identical to Game (0,0), except for during the first j queries to the CONVERT oracle, when μ^* is queried. We provide the new convert oracle used for the first j queries of Game $(0,j)$ in Figure 4.4. Let $P_{0,j}$ be the event that the adversary $\mathcal{A}$ correctly guesses b after Game $(0,j)$.

We show that Game (0,j) and Game (0, j+1) are indistinguishable assuming the DDH assumption. We provide a distinguishing algorithm $\mathcal{D}_j$ in Figure 4.5, and explain why $\mathcal{D}_j$ simulates inputs to $\mathcal{A}$ that are distributed identically to Game $(0,j)$ if a DDH tuple is input, and $\mathcal{D}_j$ simulates inputs to $\mathcal{A}$ that are distributed identically to Game $(0,j+1)$ if a DDH tuple is not input.

The values gpk, csk, isk are distributed identically to the anonymity game, as everything but h, h_1 are chosen in the same way, and χ is chosen randomly and independently, therefore h_1 is distributed correctly.

Simulating the SNDU Oracle. The SNDU oracle only differs from the oracle in the anonymity experiment during the k-th query. In this case H is distributed identically, and π_H can be simulated due to the zero-knowledge property of the proof system used. The value y is not defined, but this is not output to $\mathcal{A}$ or used in the next stage of the protocol. Therefore the outputs of SNDU are distributed identically to the anonymity experiment.

$\mathsf{SNDU}(uid, n)$

if $uid \notin \mathsf{HUL}$

$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, l \leftarrow l+1, \mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$

if $l = k \quad uid' \leftarrow uid, H \leftarrow D_2^{\chi}$, simulate π_H with $H, n, st_{uid} \leftarrow (\perp, H, \pi_H)$

return $((H, \pi_H), \mathsf{cont})$

Continue from line 5 of oracle in anonymity experiment

$\mathsf{SIGN}(uid, m)$

if $uid \neq uid'$ perform SIGN oracle from anonymity experiment

else $\alpha, \beta \leftarrow_{\$} \mathbb{Z}_p^*, \mu_1 \leftarrow g^{\alpha}, \mu_2 \leftarrow cpk^{\alpha} D_2$

$A', d \leftarrow_{\$} \mathbb{Z}_p^*, \hat{A} \leftarrow A'^{isk}$, simulate π with $A', \hat{A}, d, \mu_1, \mu_2, m$

$\sigma \leftarrow (A', \hat{A}, d, \pi) \quad$ **return** $((\mu_1, \mu_2), \sigma)$

$\mathsf{CONVERT}((\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k), bpk)$

$s \leftarrow s+1 \quad$ **if** $s \leq j$ perform $\mathsf{CONVERT}$ given in Figure 4.4

if $s > j+1$ perform $\mathsf{CONVERT}$ given in anonymity experiment

if $\exists i \in [1,k]$ s.t. $\mathsf{CLS.Verify}(gpk, m_i, \mu_i, \sigma_i) = 0 \quad$ **return** $\perp$

if $bpk \notin \mathcal{BPK} \quad$ **return** $\perp$

if $\exists i$ s.t. $\mu_i = \mu^*$ and $\exists j \neq i$ s.t. $\mathsf{Identify}(uid_d^*, \mu_j) = 1$ for $d \in \{0,1\}$

return $\perp$

else compute $(c\mu_i, c_i) \leftarrow_{\$} \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i); r_i')$ for $i = 1, \ldots, k$

if $\exists i$ s.t. $\mu_i = \mu^* \quad r \leftarrow_{\$} \mathbb{Z}_p^*$

$\forall j \in [1,k]$

$\alpha \leftarrow_{\$} \mathbb{Z}_p^*, c_j' \leftarrow (c_{j,1} g^{\alpha}, c_{j,2} bpk^{\alpha}) \quad$ Let $\sigma_j = (.,.,.,\pi_j)$

if $\mu_j = \mu^* \quad \mu_j' \leftarrow D_4^r$

if $\mu_j \neq \mu^*$

if $(uid, ., \mu_j, \sigma_j) \in \mathsf{SL} \quad y_j \leftarrow y_{uid} \quad$ **else** extract y_j from π_j

$\mu_j' \leftarrow D_3^{r y_j}$

$\beta \leftarrow_{\$} \mathbb{Z}_p^*, c\mu_j', \leftarrow (g^{\beta}, \mu_j' bpk^{\beta})$

Choose random permutation $\Pi; \forall i \in [1,k] \quad (\overline{c\mu}_i, \overline{c}_i) \leftarrow (c\mu'_{\Pi(i)}, c'_{\Pi(i)})$

else $((\overline{c\mu}_1, \overline{c}_1), \ldots, (\overline{c\mu}_k, \overline{c}_k)) \leftarrow_{\$} \mathsf{CLS.Convert}(gpk, csk, bpk, ((c\mu_1, c_1), \ldots, (c\mu_k, c_k)))$

return $(\{(c\mu_i, c_i)\}_k, \{(\overline{c\mu}_i, \overline{c}_i)\}_k, r_1', \ldots, r_k')$

$\mathcal{D}_j(D_1, D_2, D_3, D_4)$

$s, l \leftarrow 0, k \leftarrow_{\$} [1,q], b, b' \leftarrow_{\$} \{0,1\}, h \leftarrow D_1, \chi \leftarrow_{\$} \mathbb{Z}_p^*, h_1 \leftarrow h^{\chi}$

Finish computing gpk, csk, isk as in $\mathsf{CLS.Setup}, \mathsf{CLS.IKGen}, \mathsf{CLS.CKGen}$

$\mathsf{HUL}, \mathsf{SL} \leftarrow \emptyset$

$(uid_0^*, uid_1^*, m^*, \mathsf{st}) \leftarrow_{\$} \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVERT}}(gpk, isk, \mathsf{choose})$

if $uid_b^* \neq uid' \quad$ **return** 0

if $uid_0^* \notin \mathsf{HUL}$ or $uid_1^* \notin \mathsf{HUL} \quad$ **return** b'

$(\mu^*, \sigma^*) \leftarrow_{\$} \mathsf{SIGN}(uid', m^*)$

$b^* \leftarrow_{\$} \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVERT}}(\mathsf{st}, \mu^*, \sigma^*, \mathsf{guess})$

if $b^* = b \quad$ **return** 1

Figure 4.5: $\mathcal{D}_j$ a distinguishing algorithm for the DDH problem in the CLS anonymity proof

Simulating the SIGN Oracle. The SIGN oracle is identical to the oracle in the anonymity experiment, when $uid \neq uid'$ is queried. When uid' is queried, we write $\log_{D_1}(D_2)$ as $\tilde{y}$, and so $D_2 = h^{\tilde{y}}$. This is consistent with SNDU, as $H = D_2^{\chi} = h_1^{\tilde{y}}$. A', d' are chosen randomly and independently and $\hat{A} = A'^{isk}$ and so these are distributed identically to CLS.Sign. The signature proof of knowledge π can be simulated due to the zero-knowledge property of the proof system used. Therefore the outputs of SIGN are distributed identically to the anonymity experiment.

Simulating the CONVERT Oracle. Other than the jth query, the CONVERT oracle is identical to both Games $(0, j)$ and $(0, j+1)$.

For the jth query, if the input to $\mathcal{D}_j$ is a DDH tuple, then outputs from the CONVERT oracle are identically distributed to the anonymity game. This is because, if μ^* is not queried, the oracle behaves identically to the anonymity game. If μ^* is queried, the $\overline{c}_i$ are computed identically to the anonymity game. Writing $\log_{D_1}(D_3)$ as $\tilde{r}$, $D_4^r = h^{\tilde{y}\tilde{r}r}$ and $D_3^{ry_j} = h^{r\tilde{r}y_j}$. The μ' are then blinded, and shuffled with a random permutation. Therefore, due to the re-randomisation property, the freshly blinded μ' are distributed identically to the re-randomised μ'. Therefore, if a DDH tuple is input, the outputs of the CONVERT oracle are identically distributed to Game $(0, j)$.

Simulating challenge signature. The challenge signature (μ^*, σ^*) input to $\mathcal{A}$ in the guessing stage is distributed identically to the anonymity game, due to outputs of the SIGN oracle being distributed identically to the anonymity game.

Reduction to the DDH problem. If the input to $\mathcal{D}_j$ is not a DDH tuple, then outputs of the CONVERT oracle for the jth query are identically distributed to Figure 4.4. This is because for all pseudonyms μ_i input, except for μ^*, the pseudonyms output are distributed identically to the oracle in the anonymity game. When μ^* is queried, as D_4 is random and independent, μ_i' is now chosen randomly and independently, and so is identically distributed to Figure 4.4. Therefore, if the input to $\mathcal{D}_j$ is not a DDH tuple, then the outputs to $\mathcal{A}$ are identically distributed to Game $(0, j+1)$

The distinguisher $\mathcal{D}_j$ only aborts early if $uid_b^* \neq uid'$, which occurs with probability $q-1/q$.

Therefore, the probability that $\mathcal{D}_j$ outputs 1 given a DDH tuple was input is $\Pr[P_{0,j}]/q$. The probability that $\mathcal{D}_j$ outputs 1 given a DDH tuple was not input is $\Pr[P_{0,j+1}]/q$. The advantage of $\mathcal{D}_j$ is therefore $|\Pr[P_{0,j}] - \Pr[P_{0,j+1}]|/q$, therefore $|\Pr[P_{0,j}] - \Pr[P_{0,j+1}]| = q\epsilon_{\mathsf{DDH}}$.

We define Game 1 to be Game $(0, q_{\mathsf{conv}})$, where q_{conv} is the number of queries to the CONVERT oracle. Let P_1 be the event that an adversary $\mathcal{A}$ correctly guesses b after Game 1. As $|\Pr[P_{0,j}] - \Pr[P_{0,j+1}]| = q\epsilon_{\mathsf{DDH}}$, then $|\Pr[P_{0,0}] - \Pr[P_1]| \leq q_{\mathsf{conv}} q\epsilon_{\mathsf{DDH}}$.

Next, we show that $|\Pr[P_1] - 1/2| \leq \epsilon_{\mathsf{DDH}}$. We build an adversary $\mathcal{A}'$ that distinguishes between DDH tuples, given $\mathcal{A}$ that successfully guesses b in Game 1 with probability $\Pr[P_1]$. We provide $\mathcal{A}'$ in Figure 4.6, and explain why the simulation input to $\mathcal{A}$ given in Figure 4.6 is identically distributed to Game 1, provided a DDH tuple is input, and $\mathcal{A}$ guesses correctly with probability 1/2, if a DDH tuple is not input.

The group public key and issuer secret key (gpk, isk) are computed identically to Game 1, except for g, cpk, which are distributed identically to Game 1.

Simulating the Oracles. The SNDU and SIGN oracles are identical to Game 1. In the CONVERT oracle, if $\mu_j = \mu^*$, μ'_j is distributed identically to Game 1. Otherwise, μ'_i is computed using y_{uid}, which is already known if μ'_i was output by the SIGN oracle, and for other signatures y_{uid} can be extracted using the soundness property of the zero-knowledge proofs used. Due to the re-randomisation property, the freshly blinded μ' are distributed identically to the re-randomised μ'. Therefore, the $c\mu'_i$ generated are distributed identically to Figure 4.4.

Reduction to the DDH problem. If a DDH tuple is input, $(D_3, D_4 h^{y_b})$ are distributed identically to CLS.Sign. The values A', d are chosen randomly, $\hat{A} = A'^{isk}$, and so they are distributed identically to the CLS.Sign algorithm. The signature proof of knowledge π can be simulated, due to the zero-knowledge property of the zero-knowledge proofs used. Therefore, μ^*, σ^* are distributed identically to Game 1, and so $\mathcal{A}$ guesses successfully with probability $\Pr[P_1]$. If the input is not a DDH tuple, then all inputs to $\mathcal{A}$ are independent of b, therefore $\mathcal{A}$ guesses successfully with probability 1/2. On input a DDH tuple, $\mathcal{A}'$

$\mathsf{SNDU}(uid, M_{\mathsf{in}})$

Same as in anonymity experiment

$\mathsf{SIGN}(uid, m)$

Same as in anonymity experiment

$\mathsf{CONVERT}((\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k), bpk)$

if $\exists i \in [1,k]$ s.t. $\mathsf{CLS.Verify}(gpk, m_i, \mu_i, \sigma_i) = 0$ **return** $\perp$
if $bpk \notin \mathcal{BPK}$ **return** $\perp$
if $\exists i$ s.t. $\mu_i = \mu^*$ and $\exists j \neq i$ s.t. $\mathsf{Identify}(uid_d^*, \mu_j) = 1$ for $d \in \{0,1\}$
  return $\perp$
else compute $(c\mu_i, c_i) \leftarrow_\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i); r_i')$ for $i = 1, \ldots, k$
$r \leftarrow_\$ \mathbb{Z}_p^*, \forall j \in [1,k]$
  $\alpha \leftarrow_\$ \mathbb{Z}_p^*, c_j' \leftarrow (c_{j,1} g^\alpha, c_{j,2} bpk^\alpha)$
  Let $\sigma_j = (.,.,.,\pi_j)$
  if $\mu_j = \mu^*$ $\mu_j' \leftarrow_\$ \mathbb{G}_1$
  if $\mu_j \neq \mu^*$
    if $(uid, ., \mu_j, \sigma_j) \in \mathsf{SL}$ $y_j \leftarrow y_{uid}$ **else** extract y_j from σ_j
    $\mu_j' \leftarrow h^{r y_j}$
  $\beta \leftarrow_\$ \mathbb{Z}_p^*, c\mu_j', \leftarrow (g^\beta, \mu_j' bpk^\beta)$
Choose random permutation $\Pi; \forall i \in [1,k]$ $(\overline{c\mu}_i, \overline{c}_i) \leftarrow (c\mu_{\Pi(i)}', c_{\Pi(i)}')$
return $(\{(c\mu_i, c_i)\}_k, \{(\overline{c\mu}_i, \overline{c}_i)\}_k, r_1', \ldots, r_k')$

$\mathcal{A}'(D_1, D_2, D_3, D_4)$

Set $g \leftarrow D_1, b, b' \leftarrow_\$ \{0,1\}, cpk \leftarrow D_2$
Otherwise set gpk, isk as in $\mathsf{CLS.Setup}, \mathsf{CLS.IKGen}, \mathsf{CLS.CKGen}$
$(uid_0^*, uid_1^*, m^*, \mathsf{st}) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVERT}}(\mathsf{choose}, gpk, isk)$
Let $\mathbf{gsk}[uid_0^*] = (\cdot, \cdot, y_0, \cdot)$ and $\mathbf{gsk}[uid_1^*] = (\cdot, \cdot, y_1, \cdot)$, **if** y_0, y_1 undefined **return** b'
$\mu_1^* \leftarrow D_3, \mu_2^* \leftarrow D_4 h^{y_b}$
$A', d \leftarrow_\$ \mathbb{G}_1, \hat{A} \leftarrow A'^{isk}$, simulate π with $A', \hat{A}, d, \mu_1^*, \mu_2^*, m^*$
$\mu^* \leftarrow (\mu_1^*, \mu_2^*), \sigma^* \leftarrow (A', \hat{A}, d, \pi), b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVERT}}(\mathsf{guess}, \mathsf{st}, \mu^*, \sigma^*)$
if $b^* = b$ **return** 1 **else return** 0

Figure 4.6: $\mathcal{A}'$ which distinguishes between DDH tuples using $\mathcal{A}$ in the CLS anonymity proof

outputs 1 with probability $\Pr[P_1]$. On an input that is not a DDH tuple, $\mathcal{A}'$ outputs 1 with probability 1/2. Therefore $\Pr[P_1] - 1/2 \leq \epsilon_{\mathsf{DDH}}$, and so $|\Pr[P_1] - 1/2| = \epsilon_{\mathsf{DDH}}$.

Therefore $|\Pr[P_{0,0}] - 1/2| \leq (qq_{\mathsf{conv}} + 1)\epsilon_{\mathsf{DDH}}$. As ϵ_{DDH} is negligible, our construction satisfies anonymity.

□

4.4.3 Non–transitivity

Lemma 4.2. The CLS–DDH construction presented in Section 4.3.1 satisfies **non-transitivity** if the DDH assumption holds in $\mathbb{G}_1$, and the SPK is unbounded simulation-sound, zero–knowledge and online extractable (for the underlined value).

Proof. For proving non-transitivity, we have to show that there exists an efficient simulator SIM that makes the real and simulated game indistinguishable. We start by describing the simulator and then explain why the real and simulated conversion oracles CONVERT and CONVSIM are indistinguishable.

$\mathsf{SIM}(gpk, bpk, \mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k'}})$

$l \leftarrow 0, \forall j \in [1, k']$

$\mu' \leftarrow_{\$} \mathbb{G}_1; \forall m \in L_{uid_j}$

$l \leftarrow l + 1, \beta, \gamma \leftarrow_{\$} \mathbb{Z}_p^*, \overline{c}_l \leftarrow (g^\gamma, m \cdot bpk^\gamma), \overline{c\mu}_l \leftarrow (g^\beta, \mu' \cdot bpk^\beta)$

return $((\overline{c\mu}_1, \overline{c}_1), \ldots (\overline{c\mu}_l, \overline{c}_l))$

We assume that an adversary $\mathcal{A}$ exists that makes q queries to the SNDU oracle for distinct user identifiers and q_{conv} queries to the CONVX oracle, guesses b correctly in the non-transitivity game with SIM as described and wins with probability $\epsilon + 1/2$.

We will stepwise make the real-world (b=0) and the simulated world (b=1) equivalent, using a sequence of Games $\mathcal{H}_j$ for $j = 0, \ldots, q$. The idea is that in Game $\mathcal{H}_j$ we will not use simulated conversions for all users $uid_1, \ldots, uid_j$ in order of when they were queried

to SNDU. More precisely, we define Game $\mathcal{H}_j$ to be as given in Figure 4.7 with all other oracles identical to the non–transitivity experiment. Let P_j be the event that $\mathcal{A}$ guesses b correctly in Game $\mathcal{H}_j$, with the simulator given. Game $\mathcal{H}_j$ keeps track of the queries to SNDU, adding the first j queries uid to a set UL. Then during queries to CONVSIM, if a signature of a user in UL is queried, these are treated in the same way as pseudonyms for corrupted users, i.e., they are normally converted using the CLS.Convert algorithm. If a signature of an honest user that is not in UL is queried, we add this user to NUL. These conversions are simulated as usual.

Game $\mathcal{H}_0$ is identical to the non-transitivity game because UL is empty. Therefore, $\Pr[P_0] = \epsilon + 1/2$. In Game $\mathcal{H}_q$, UL contains all honest users, and so the CONVSIM oracle is now identical to the CONVERT oracle, and inputs to the adversary are now independent of b , therefore $\Pr[P_q] = 1/2$.

We now show that if an adversary can distinguish Games $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$, we can turn this into a distinguisher $\mathcal{D}_j$ that can break the DDH assumption. We describe the reduction and the additional simulation that is needed in Figures 4.8 and 4.9. The CONVERT oracle remains unchanged. To avoid confusion, we write uid' to refer to the $j+1$-th user that has joined the group (and for which $\mathcal{D}_j$ embedded the DDH challenge).

We now argue that when a DDH tuple (D_1, D_2, D_3, D_4) is input to $\mathcal{D}_j$, the inputs to $\mathcal{A}$ are distributed identically to Game $\mathcal{H}_{j+1}$; when a DDH tuple is not input, the inputs to $\mathcal{A}$ are distributed identically to Game $\mathcal{H}_j$. That is for $D_1 = h, D_2 = h^a, D_3 = h^b, D_4 = h^c$, the oracles provided by $\mathcal{D}_j$ will be exactly as in $\mathcal{H}_{j+1}$ when $c = ab$, and as in $\mathcal{H}_j$ otherwise.

We first note that due to the DDH random self-reduction given in [127], if a DDH tuple is input to $\mathcal{D}_j$, then for all $i \in [q_{\mathsf{conv}}]$, $D_1, D_2, D_{3,i}, D_{4,i}$ is a DDH tuple. If a DDH tuple is not input to $\mathcal{D}_j$, then $D_1, D_2, D_{3,1}, D_{4,1}, \cdots, D_{3,q_{\mathsf{conv}}}, D_{4,q_{\mathsf{conv}}}$ is randomly and independently distributed.

The keys gpk, csk, isk are distributed identically to the non–transitivity game, as χ is chosen randomly and independently when setting $h_1 \leftarrow h^{\chi}$.

Simulating the SNDU Oracle. The SNDU oracle only differs from the oracle in the non–transitivity experiment during the $(j+1)$-th query by embedding D_2 of the DDH chal-

Game $\mathcal{H}_j$

$t \leftarrow 0, b \leftarrow_\$ \{0,1\}, \mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau)$
$\mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \emptyset$
$(ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk), b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVX}}(\mathsf{guess}, gpk, isk)$
return b^*

$\mathsf{SNDU}(uid, M_{\mathsf{in}})$

if $uid \notin \mathsf{HUL}, t \leftarrow t+1,$ **if** $t \leq j \quad \mathsf{UL} \leftarrow \mathsf{UL} \cup \{uid\}$
Continue from line 2 of standard SNDU oracle

$\mathsf{CONVSIM}((\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k), bpk)$

if $\exists i \in [1,k]$ s.t. $\mathsf{CLS.Verify}(gpk, m_i, \mu_i, \sigma_i) = 0$ or $bpk \notin \mathcal{BK}$ **return** $\perp$
Set $\mathsf{CL} \leftarrow \emptyset$
Compute $(c\mu_i, c_i) \leftarrow_\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i); r_i)$ for $i = 1, \ldots, k$
$\forall i \in [1,k]$
  if $(uid, m_i, \mu_i, \sigma_i) \in \mathsf{SL}$
    if L_{uid} does not exist, create $\mathsf{L}_{uid} \leftarrow \{m_i\}, \mathsf{CL}_{uid} \leftarrow \{(c_i, c\mu_i)\}$
    else set $\mathsf{L}_{uid} \leftarrow \mathsf{L}_{uid} \cup \{m_i\}, \mathsf{CL}_{uid} \leftarrow \mathsf{CL}_{uid} \cup \{(c_i, c\mu_i)\}$
  else $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(c_i, c\mu_i)\}$
$\{(\overline{c\mu}_i, \overline{c}_i)\}_{i=1,\ldots k'} \leftarrow_\$ \mathsf{CLS.Convert}(gpk, csk, bpk, \mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid})$
for $k' \leftarrow |\mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid}|$
let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters
Let $\mathsf{NUL} \leftarrow \{uid_1, \ldots uid_{k''}\} \backslash \mathsf{UL}$
$\{(\overline{c\mu}_i, \overline{c}_i)\}_{i=k'+1,\ldots k} \leftarrow_\$ \mathsf{SIM}(gpk, bpk, \bigcup_{uid \in \mathsf{NUL}} \mathsf{L}_{uid})$
Choose random permutation $\Pi; \forall i \in [1,k] \quad (\overline{c\mu}'_i, \overline{c}'_i) \leftarrow (\overline{c\mu}_{\Pi(i)}, \overline{c}_{\Pi(i)})$
return $(\{(c\mu_i, c_i, r_i)\}_{i=1,\ldots,k}, \{(\overline{c\mu}'_i, \overline{c}'_i)\}_{i=1,\ldots k}, r_1, \ldots, r_k)$

Figure 4.7: Description of Game $\mathcal{H}_j$ and the changes to the SNDU and CONVSIM oracles in the CLS non–transitivity proof

$\mathcal{D}_j(D'_1, D'_2, D'_3, D'_4)$

Expand D'_1, D'_2, D'_3, D'_4 into $D_1, D_2, \{D_{3,i}, D_{4,i} : i \in [q_{\mathsf{conv}}]\}$ using DDH random self-reduction [127]
$s, t \leftarrow 0, b, \leftarrow\!\!\$\, \{0,1\}, h \leftarrow D_1, \chi \leftarrow\!\!\$\, \mathbb{Z}_p^*, h_1 \leftarrow h^\chi$
Finish computing gpk, csk, isk as in CLS.Setup, CLS.IKGen, CLS.CKGen
HUL, CUL, SL $\leftarrow \emptyset$
$b^* \leftarrow\!\!\$\, \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVX}}(\mathsf{guess}, gpk, isk)$
if $b^* = b$ **return** 1 **else return** 0

SNDU(uid, n)

if $uid \notin$ HUL, $t \leftarrow t+1$, **if** $t \leq j$ UL $\leftarrow$ UL $\cup \{uid\}$
HUL $\leftarrow$ HUL $\cup \{uid\}$, $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow$ cont
if $t = j+1$ $uid' \leftarrow uid, H \leftarrow D_2^\chi$ simulate π_H with $H, n, st_{uid} \leftarrow (\perp, H, \pi_H)$
return $((H, \pi_H), \mathsf{cont})$
Continue from line 5 of standard SNDU oracle

SIGN(uid, m)

if $uid \neq uid'$ perform SIGN oracle from anonymity experiment
else $\alpha, \beta \leftarrow\!\!\$\, \mathbb{Z}_p^*, \mu_1 \leftarrow g^\alpha, \mu_2 \leftarrow cpk^\alpha D_2$
$A', d \leftarrow\!\!\$\, \mathbb{Z}_p^*, \hat{A} \leftarrow A'^{isk}$
Simulate π with $A', \hat{A}, d, \mu_1, \mu_2, m$
$\sigma \leftarrow (A', \hat{A}, d, \pi)$ **return** $((\mu_1, \mu_2), \sigma)$

Figure 4.8: Oracles for $\mathcal{D}_j$ our distinguishing algorithm for the DDH problem in the CLS non–transitivity proof

lenger into the user's "public key" H using knowledge of χ. Clearly, H is distributed identically as when computed normally, and π_H can be simulated due to the zero-knowledge property of the proof system. Note that y is not defined for this honest user, but this is not output to $\mathcal{A}$, or used in the next stage of the protocol.

Simulating the SIGN Oracle. The SIGN oracle is identical to the oracle in the non–transitivity experiment when $uid \neq uid'$ is queried. When uid' is queried, we simply encrypt D_2 instead of h^y.

This is consistent with SNDU, as $H = D_2^\chi$. Further, A', d' are chosen randomly and independently, and $\hat{A} = A'^{isk}$, so these are distributed identically to CLS.Sign. The SPK π can be simulated due to the zero–knowledge property of the proof system.

Simulating the CONVSIM Oracle. What remains to be shown is that the CONVSIM oracle created by $\mathcal{D}_j$ either behaves identically to the CONVSIM oracle in Game $\mathcal{H}_j$ or as in $\mathcal{H}_{j+1}$, depending on whether its input was a DDH tuple or not. We know that $D_{3,s} = h^{\tilde{r}}$ for some $\tilde{r}$ and thus it must hold that $D_{3,s}^{yr} = h^{\tilde{r}ry}$. Finally, we derive $\overline{c\mu}$ by

$\mathsf{CONVSIM}((\mu_1, m_1, \sigma_1), \ldots, (\mu_k, m_k, \sigma_k), bpk)$

$s \leftarrow s+1$, **if** $\exists i \in [1,k]$ s.t. $\mathsf{CLS.Verify}(gpk, m_i, \mu_i, \sigma_i) = 0$ or $bpk \notin \mathcal{BK}$ **return** $\perp$
Set $\mathsf{CL} \leftarrow \emptyset, r \leftarrow_\$ \mathbb{Z}_p^*$
Compute $(c\mu_i, c_i) \leftarrow_\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_i, m_i); r_i)$ for $i = 1, \ldots, k$
$\forall i \in [1,k]$
  if $(uid, m_i, \mu_i, \sigma_i) \in \mathsf{SL}$
    if L_{uid} does not exist $\mathsf{L}_{uid} \leftarrow \{m_i\}, \mathsf{CL}_{uid} \leftarrow \{(m_i, y_{uid})\}$
    else $L_{uid} \leftarrow \mathsf{L}_{uid} \cup \{m_i\}, \mathsf{CL}_{uid} \leftarrow \mathsf{CL}_{uid} \cup \{(m_i, y_{uid})\}$
  else Extract y_i from $\sigma_i, \mathsf{CL} \leftarrow \mathsf{CL} \cup \{(m_i, y_i)\}$
$n \leftarrow 0$; $\forall (m,y) \in \mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid}$
  $n \leftarrow n+1, \gamma_1, \gamma_2 \leftarrow_\$ \mathbb{Z}_p^*, (\overline{c\mu}_n, \overline{c}_n) \leftarrow ((g^{\gamma_1}, D_{3,s}^{yr} bpk^{\gamma_1}), (g^{\gamma_2}, mbpk^{\gamma_2}))$
if $\mathsf{L}_{uid'}$ exists $\forall m \in L_{uid'}$
  $n \leftarrow n+1, \gamma_1, \gamma_2 \leftarrow_\$ \mathbb{Z}_p^*, (\overline{c\mu}_n, \overline{c}_n) \leftarrow ((g^{\gamma_1}, D_{4,s}^{r} bpk^{\gamma_1}), (g^{\gamma_2}, mbpk^{\gamma_2}))$
let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters
Let $\mathsf{NUL} \leftarrow \{uid_1, \ldots uid_{k''}\} \backslash \mathsf{UL}$
$\{(\overline{c\mu}_i, \overline{c}_i)\}_{i=n+1,\ldots k} \leftarrow_\$ \mathsf{SIM}(gpk, bpk, \bigcup_{uid \in \mathsf{NUL}, uid \neq uid'} \mathsf{L}_{uid})$
Choose random permutation $\Pi; \forall i \in [1,k]$ $(\overline{c\mu}'_i, \overline{c}'_i) \leftarrow (\overline{c\mu}_{\Pi(i)}, \overline{c}_{\Pi(i)})$
return $(\{(c\mu_i, c_i)\}_{i=1,\ldots,k}, \{(\overline{c\mu}'_i, \overline{c}'_i)\}_{i=1,\ldots k}, r_1, \ldots, r_k)$

Figure 4.9: The CONVSIM oracle used by distinguisher $\mathcal{D}_j$ in the CLS non–transitivity proof

encrypting $D_{3,s}^{yr}$ from scratch under bpk, which is not noticeable to the adversary due to the re-randomisation that is applied in the conversion algorithm.

If (D_1, D_2, D_3, D_4) is a DDH tuple, then $D_{4,s}^r = h^{\tilde{r}r\tilde{y}}$. Therefore as $\tilde{y} = y_{uid'}$, the inputs to $\mathcal{A}$ are also distributed identically to Game $\mathcal{H}_{j+1}$. Whereas, if (D_1, D_2, D_3, D_4) is *not* a DDH tuple, then $D_{4,s}^r$, is distributed identically to μ', which was chosen randomly and independently. Therefore the inputs to $\mathcal{A}$ are distributed identically to Game $\mathcal{H}_j$.

Reduction to the DDH problem. Therefore, the probability that $\mathcal{D}_j$ outputs 1 if it was given a valid DDH tuple as input is $\Pr[P_{j+1}]$, and $\Pr[P_j]$ is the probability that $\mathcal{D}_j$ outputs 1 when the input was not a DDH tuple. The advantage of $\mathcal{D}_j$ is then $|\Pr[P_j] - \Pr[P_{j+1}]|$, therefore $|\Pr[P_j] - \Pr[P_{j+1}]| = \epsilon_{\mathsf{DDH}}$.

Overall, for our sequence of games $\mathcal{H}_0$ to $\mathcal{H}_q$, it holds that $|\Pr[P_0] - \Pr[P_q]| \leq q\epsilon_{\mathsf{DDH}}$ and thus $\epsilon \leq q\epsilon_{\mathsf{DDH}}$ is negligible. This concludes our proof that the CLS–DDH construction satisfies non–transitivity. □

4.4.4 Conversion Blindness

Lemma 4.3. The CLS–DDH construction presented in Section 4.3.1 satisfies **conversion blindness** if the DDH assumption holds in $\mathbb{G}_1$.

Proof. We define Game 0 be the conversion blindness experiment, using the CLS–DDH construction. Let P_0 be the event that an adversary $\mathcal{A}$ correctly guesses b after Game 0.

We define Game 1 to be identical to Game 0, except instead $c\mu^*$ is chosen randomly. Let P_1 be the event that the adversary $\mathcal{A}$ correctly guesses b after Game 1.

Game 1

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), (bpk, bsk) \leftarrow_\$ \mathsf{CLS.BKGen}(\mathsf{param})$

$(\mathsf{st}, (\mu_0, m_0), (\mu_1, m_1)) \leftarrow_\$ \mathcal{A}(\mathsf{choose}, gpk, isk, csk, bpk)$

$(., c^*) \leftarrow_\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_b, m_b)), c\mu^* \leftarrow_\$ \mathbb{G}_1^3$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, c\mu^*, c^*)$

return 1 **if** $b^* = b$

We define Game 2 to be identical to Game 1, except c^* is chosen randomly. Let P_2 be the event that the adversary $\mathcal{A}$ correctly guesses b after Game 2. Clearly the probability that $\mathcal{A}$ correctly guesses b is $1/2$, as they are given no information about b. Therefore $\Pr[P_2] = 1/2$.

Game 2

$\mathsf{param} \leftarrow_\$ \mathsf{CLS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), (bpk, bsk) \leftarrow_\$ \mathsf{CLS.BKGen}(\mathsf{param})$

$(\mathsf{st}, (\mu_0, m_0), (\mu_1, m_1)) \leftarrow_\$ \mathcal{A}(\mathsf{choose}, gpk, isk, csk, bpk)$

$c^* \leftarrow_\$ \mathbb{G}_1^2, c\mu^* \leftarrow_\$ \mathbb{G}_1^3$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, c\mu^*, c^*)$

return 1 **if** $b^* = b$

We show that Game 0 and Game 1 are indistinguishable assuming the DDH assumption. We provide a distinguishing algorithm $\mathcal{D}_1$ in Figure 4.10.

$\mathcal{D}_1(D_1, D_2, D_3, D_4)$

$b \leftarrow\!\!\$ \{0,1\}$
$g \leftarrow D_1, bpk \leftarrow D_2$, otherwise generate gpk, isk, csk as in $\mathsf{CLS.Setup}, \mathsf{CLS.IKGen}, \mathsf{CLS.CKGen}$
$(\mathsf{st}, ((\mu_0, \mu_0'), m_0), ((\mu_1, \mu_1'), m_1)) \leftarrow\!\!\$ \mathcal{A}(\mathsf{choose}, gpk, isk, csk, bpk)$
$(., c^*) \leftarrow\!\!\$ \mathsf{CLS.Blind}(gpk, bpk, (\mu_b, m_b)), \beta \leftarrow\!\!\$ \mathbb{Z}_p^*, c\mu^* \leftarrow (\mu_b g^\beta, D_3, \mu_b' D_4 cpk^\beta)$
$b^* \leftarrow\!\!\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, c\mu^*, c^*)$
return 1 **if** $b^* = b$

Figure 4.10: $\mathcal{D}_1$ that distinguishes between Game 0 and Game 1 in the CLS conversion blindness proof

$\mathcal{D}_2(D_1, D_2, D_3, D_4)$

$b \leftarrow\!\!\$ \{0,1\}$
$g \leftarrow D_1, bpk \leftarrow D_2$, otherwise generate gpk, isk, csk as in $\mathsf{CLS.Setup}, \mathsf{CLS.IKGen}, \mathsf{CLS.CKGen}$
$(\mathsf{st}, ((\mu_0, \mu_0'), m_0), ((\mu_1, \mu_1'), m_1)) \leftarrow\!\!\$ \mathcal{A}(\mathsf{choose}, gpk, isk, csk, bpk)$
$c^* \leftarrow (D_3, m_b D_4), c\mu^* \leftarrow\!\!\$ \mathbb{G}_1^3$
$b^* \leftarrow\!\!\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, c\mu^*, c^*)$
return 1 **if** $b^* = b$

Figure 4.11: $\mathcal{D}_2$ that distinguishes between Game 1 and Game 2 in the CLS conversion blindness proof

If $\mathcal{D}_1$ is input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 0. This is because, letting $\alpha = \log_g D_3$, then $D_4 = bpk^\alpha$, and therefore $c\mu^*$ is distributed identically to the CLS.Blind algorithm.

If $\mathcal{D}_1$ is not input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 1. This is because, β, D_3, D_4 are now chosen independently and randomly.

Therefore $|\Pr[P_0] - \Pr[P_1]| \leq \epsilon_{\mathsf{DDH}}$, where ϵ_{DDH} is the DDH advantage, and therefore negligible.

We show that Game 1 and Game 2 are indistinguishable assuming the DDH assumption. We provide a distinguishing algorithm $\mathcal{D}_2$ in Figure 4.11.

If $\mathcal{D}_2$ is input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 1. This is because letting $\alpha = \log_g D_3$, then $D_4 = bpk^\alpha$, and so c^* is distributed identically to an output of the CLS.Blind algorithm.

If $\mathcal{D}_2$ is not input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 2. This is because D_3, D_4 are now chosen independently and randomly.

Therefore $|\Pr[P_1] - \Pr[P_2] \leq \epsilon_{\mathsf{DDH}}$, and so$|\Pr[P_0] - 1/2| \leq 2\epsilon_{\mathsf{DDH}}$.

□

4.4.5 Join Anonymity

Lemma 4.4. The CLS–DDH construction presented in Section 4.3.1 satisfies **join anonymity** if the DDH assumption holds in $\mathbb{G}_1$, and the SPK is zero–knowledge.

Proof. First we show that if an adversary $\mathcal{A}$ exists that makes q queries for distinct user identifiers to the SNDU oracle, such that

$$|\Pr[\mathbf{Exp}^{anon-join-0}_{\mathcal{A},\mathsf{CLS–DDH}}(\tau) = 1] - \Pr[\mathbf{Exp}^{anon-join-1}_{\mathcal{A},\mathsf{CLS–DDH}}(\tau) = 1]| = \epsilon,$$

where ϵ is non-negligible, then we can can build an adversary $\mathcal{A}'$,that breaks the DDH assumption with non-negligible probability. We provide $\mathcal{A}'$ in Figure 4.12. We then describe why the simulation given in Figure 4.12 is indistinguishable to the join anonymity experiment to $\mathcal{A}$ if a DDH tuple is input, and that $\mathcal{A}$ guesses correctly with probability 1/2 if a DDH tuple is not input.

Assuming the input to $\mathcal{A}'$ is not a DDH tuple, then the probability $\mathcal{A}'$ aborts due to $uid_b \neq \tilde{uid}$ is (q-1)/q. If $\tilde{uid}$ is input to the SIGN oracles, then $\mathcal{A}'$ outputs 1 with probability 1/2. As D_4 is random and independent to all other variables, all inputs to $\mathcal{A}$ are independent of b, therefore the probability they guess correctly is 1/2. Therefore, the probability that $\mathcal{A}'$ outputs 1 is $1/2q$.

Assuming the input to $\mathcal{A}'$ is a DDH tuple, $\tilde{uid} = uid_b$, and $\mathcal{A}$ does not cheat by querying uid_0 or uid_1 to the signing oracle, we now show that all inputs to $\mathcal{A}$ are distributed identically to the join anonymity experiment.

The keys (gpk, isk, csk) given to $\mathcal{A}$ are distributed identically to the join anonymity experiment. The only difference is in choosing h_1, h, which are distributed randomly and independently.

SNDU(uid, n)

if $uid \notin \mathsf{HUL}$

$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, l \leftarrow l + 1, \mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$

if $l = k \quad \tilde{uid} \leftarrow uid, H \leftarrow D_2,$ simulate π_H with $H, n, st_{uid} \leftarrow (\perp, H, \pi_H)$

return $((H, \pi_H), \mathsf{cont})$

Continue from line 5 of oracle in second converter anonymity experiment

SIGN(uid, m)

if $uid \notin \mathsf{HUL}$ **return** $\perp$

if $uid = \tilde{uid}$ **return** $\perp$

if $uid \neq uid^*$ perform SIGN oracle from join anonymity

else $\alpha, \beta \leftarrow_\$ \mathbb{Z}_p^*, \mu_1 \leftarrow g^\alpha, \mu_2 \leftarrow cpk^\alpha D_4$

$A', d \leftarrow_\$ \mathbb{G}_1, \hat{A} \leftarrow A'^{isk},$ simulate π with $A', \hat{A}, d, \mu_1, \mu_2, m$

$\sigma \leftarrow (A', \hat{A}, d, \pi)$ **return** $((\mu_1, \mu_2), \sigma)$

$\mathcal{A}'(D_1, D_2, D_3, D_4)$

$b \leftarrow_\$ \{0, 1\}, gpk, csk, isk$ chosen as in CLS.Setup, CLS.IKGen, CLS.CKGen except $h_1 \leftarrow D_1, h \leftarrow D_3$

$k \leftarrow_\$ [1, q], l = 0, \mathsf{HUL} \leftarrow \emptyset$

$(\mathsf{st}, uid_0, uid_1) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(\mathsf{choose}, gpk, isk, csk)$

if uid_0 or $uid_1 \notin \mathsf{HUL}$ or $\mathbf{dec}^{uid_0} \neq \mathsf{accept}$ or $\mathbf{dec}^{uid_1} \neq \mathsf{accept} \quad b^* \leftarrow 0$

if $b^* = b$ **return** 1 **else return** 0

if $uid_b \neq \tilde{uid}$ **return** 0

Choose uid^* as in experiment, $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid^*\}, b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(\mathsf{guess}, \mathsf{st}, uid^*)$

if $(uid_d, *) \in \mathsf{SL}$ for $d = 0, 1 \quad b^* \leftarrow 0$

if $b^* = b$ **return** 1 **else return** 0

Figure 4.12: $\mathcal{A}'$, which breaks the DDH assumption, using $\mathcal{A}$, which breaks the join anonymity of CLS–DDH with probability ϵ

Simulating the SNDU Oracle. The SNDU oracle only differs from the oracle in the join anonymity experiment during the k-th query. In this case H is distributed identically, and π_H can be simulated due to the zero-knowledge property of the proof system used. The value $y_{\tilde{uid}}$ is not defined but this is not output to $\mathcal{A}$, or used in the next stage of the protocol. Therefore, the output of SNDU is distributed identically to the join anonymity experiment.

Simulating the SIGN Oracle. The SIGN oracle is identical to the oracle in the join anonymity experiment when $uid \neq uid^*$ is queried. When uid^* is queried, letting $y^* = \log_{D_1}(D_2)$, $D_4 = h^{y^*}$. This is consistent with SNDU, as $H = D_2 = h_1^{y^*}$. The values A', d' are chosen randomly and independently, and $\hat{A} = A'^{isk}$, so these are distributed identically to CLS.Sign. The proof π can be simulated due to the zero-knowledge property of the proof system used. Therefore the output of SIGN is distributed identically to the join anonymity experiment.

Reduction to the DDH problem. Assume $\mathcal{A}'$ is input a DDH tuple, and uid_b output by $\mathcal{A}$ is $\tilde{uid}$, which occurs with probability $1/q$, then $\mathcal{A}'$ will not abort early. If the adversary queries the signing oracle with uid_0 or uid_1, or does not output honestly joined users, then $b^* = 0$ is the same as the output of the join anonymity experiment with adversary $\mathcal{A}$. If the adversary does not query the signing oracle with uid_0 or uid_1, then all inputs to $\mathcal{A}$ are identically distributed to the join anonymity experiment, and so $b^* = b$ with the same probability as 1 is output in the join anonymity experiment for adversary $\mathcal{A}$.

Therefore, the probability that $\mathcal{A}'$ outputs 1 is:

$$1/2\Pr[\mathbf{Exp}^{anon-join-0}_{\mathcal{A},\mathsf{CLS–DDH}}(\tau) = 0] + 1/2\Pr[\mathbf{Exp}^{anon-join-1}_{\mathcal{A},\mathsf{CLS–DDH}}(\tau) = 1] = (\epsilon + 1)/2.$$

Therefore $\mathcal{A}'$ outputs 1 with probability $(\epsilon + 1)/2q$.

Therefore, $\mathcal{A}'$ has advantage $\epsilon/2q$ in distinguishing DDH tuples, and so our CLS–DDH construction satisfies join anonymity.

□

$\mathsf{SNDU}(uid, M_{\mathsf{in}})$

if $uid \notin \mathsf{HUL}$

$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$

$y_{uid} \leftarrow_{\$} \mathbb{Z}_p^*, H \leftarrow D_2^{y_{uid}}$, simulate π_H with $H, st_{uid} \leftarrow (\perp, H, \pi_H)$

return $((H, \pi_H), \mathsf{cont})$

Continue from line 5 of oracle in non-frameability experiment

$\mathsf{SIGN}(uid, m)$

$\alpha \leftarrow_{\$} \mathbb{Z}_p^*, \mu_1 \leftarrow g^{\alpha}, \mu_2 \leftarrow cpk^{\alpha} D_2^{z y_{uid}}$

$A', d \leftarrow_{\$} \mathbb{G}_1, \hat{A} \leftarrow A'^{isk}$, simulate π with $A', \hat{A}, d, \mu_1, \mu_2, m$

$\sigma \leftarrow (A', \hat{A}, d, \pi)$ **return** $((\mu_1, \mu_2), \sigma)$

$\mathcal{A}'(D_1, D_2)$

gpk, csk, isk chosen as in $\mathsf{CLS.Setup}, \mathsf{CLS.IKGen}, \mathsf{CLS.CKGen}$ except $h_1 \leftarrow D_1, z \leftarrow_{\$} \mathbb{Z}_p^*, h \leftarrow D_1^z$

$\mathsf{HUL} \leftarrow \emptyset, (uid^*, m^*, \mu^*, \sigma^*) \leftarrow_{\$} \mathcal{A}^{\mathsf{SNDU}, \mathsf{SIGN}}(gpk, isk, csk)$

Extract y^* from π^* included in σ^* **return** $y^* y_{uid^*}^{-1}$

Figure 4.13: $\mathcal{A}'$, which breaks the discrete log assumption, using $\mathcal{A}$, which breaks the non-frameability of CLS–DDH with probability ϵ

4.4.6 Non–frameability

Lemma 4.5. The CLS–DDH construction presented in Section 4.3.1 satisfies **non-frameability** if the DL assumption holds in $\mathbb{G}_1$, and the SPK is simulation-sound and zero–knowledge.

Proof. We show that if there exists an adversary $\mathcal{A}$ such that $\Pr[\mathbf{Exp}^{nonframe}_{\mathcal{A},\mathsf{CLS–DDH}}(\tau) = 1] = \epsilon$, where ϵ is non-negligible, then we can can build an adversary $\mathcal{A}'$ that breaks the discrete log assumption with non-negligible probability. We provide $\mathcal{A}'$ in Figure 4.13, where we describe how $\mathcal{A}'$ simulates the oracles needed in the non-frameability game and extracts the DL solutions. We then describe why the simulation given in Figure 4.13 and the non–frameability experiment are indistinguishable towards $\mathcal{A}$.

All inputs to $\mathcal{A}$ are distributed identically to the non-frameability experiment.

The values gpk, csk, isk are distributed identically to the non–frameability game, as everything but h, h_1 is chosen in the same way. The value z is chosen randomly and independently, therefore h is distributed correctly.

Simulating the SNDU Oracle. When a new user identifier is queried to SNDU, H is distributed identically as y_{uid} is chosen randomly and independently, and π_H can be simulated due to the zero-knowledge property of proof system used. Note that y is set as $\perp$, but this is not output to $\mathcal{A}$, or used in the next stage of the protocol.

Simulating the SIGN Oracle. Letting $\tilde{y} = \log_{D_1}(D_2)$, then $D_2^{zy_{uid}} = h^{\tilde{y}y_{uid}}$. This is consistent with SNDU because $H = h_1^{\tilde{y}y_{uid}}$. Therefore, μ_1, μ_2 are distributed correctly. $A', \hat{A}, d$ are distributed identically to CLS.Sign because $\hat{A} = A'^{isk}$, and A', d are chosen randomly. The proof π can be simulated due to the zero–knowledge property of the zero–knowledge proofs used. Therefore the output of SIGN is distributed identically to the non-frameability experiment.

Reduction to the DL problem. We assume $\mathcal{A}$ is successful. The output of $\mathcal{A}$, (m^*, μ^*, σ^*), cannot have been returned by the SIGN oracle, as $\mathcal{A}$ is successful, therefore we can extract y^*, due to the soundness property of the zero–knowledge proof used. As $\mathsf{Identify}(gpk, csk, uid^*, m^*, \mu^*) = 1$, this means that $\mu_2^* {\mu^*}_1^{-csk} = D_2^{zy_{uid^*}} = h^{\tilde{y}y_{uid^*}}$. Therefore due to the soundness of the zero–knowledge proof, $h^{\tilde{y}y_{uid^*}} = h^{y^*}$ and $\tilde{y}$ is output by $\mathcal{A}'$.

If $\mathcal{A}$ is successful with probability ϵ, $\mathcal{A}'$ solves the discrete logarithm problem with probability at least ϵ, and so our CLS–DDH construction satisfies non–frameability.

□

4.4.7 Traceability

Lemma 4.6. The CLS–DDH construction presented in Section 4.3.1 satisfies **traceability** if the q-SDH assumption holds, and the SPK is simulation-sound, zero–knowledge and online extractable.

Proof. First we show that if there exists an adversary $\mathcal{A}$ such that $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CLS–DDH}}(\tau) = 1] = \epsilon$, where ϵ is non-negligible, which makes q queries to the SNDI oracle for distinct user identifiers, then we can can build an adversary $\mathcal{A}'$ that breaks the q-SDH problem

with non-negligible probability. We provide the detailed description of $\mathcal{A}'$ in Figure 4.14, and explain here how $\mathcal{A}'$ works.

First note that all inputs that $\mathcal{A}'$ provides to $\mathcal{A}$ are distributed identically to the traceability experiment. This is because $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_2, ipk$ are distributed identically to the traceability experiment, and g, h, cpk, csk were chosen identically to the traceability experiment. The generators g_1, h_2 are distributed correctly, due to the fact μ and ν_1 were chosen randomly and independently. As θ is chosen randomly, h_1 is also distributed correctly. Therefore, (gpk, csk) given to $\mathcal{A}$ are distributed identically to the traceability experiment.

We will later use the fact that $\Gamma = g_1^{isk}$, and $B_i = g_1^{1/(isk+x_i)}$. This is because $g_1 = \prod_{i=0}^{q-1} S_i^{\mu\zeta_i} = \tilde{g}_1^{\mu f(isk)}$, therefore $\Gamma = \prod_{i=0}^{q-1} S_{i+1}^{\mu\zeta_i} = \prod_{i=0}^{q-1} S_i^{\mu\zeta_i isk} = g_1^{isk}$, and $B_i = \prod_{j=0}^{q-2} S_j^{\mu\eta_{i,j}}$ $= \tilde{g}_1^{\mu f_i(isk)} = g_1^{1/(isk+x_i)}$.

Simulating the ADDU Oracle. The ADDU oracle simply returns accept when a valid uid is input, as in the traceability experiment.

Simulating the SNDI Oracle. In the case of SNDI, y can be extracted due to the soundness property of the zero-knowledge proofs used. We now show that answers to SNDI queries are correctly distributed.

During the kth query to SNDI, s_{uid}, x_{uid} are distributed correctly, as ν_2 is chosen randomly and independently. Because $\nu_2 = s_{uid} + \theta y$,

$$A_{uid} = g_1^{\nu_1} = (g_1 g_1^{\nu_1(isk+x)-1})^{1/(isk+x)} = (g_1 h_2^{\nu_2})^{1/(isk+x)} = (g_1 h_2^{s_{uid}+\theta y})^{1/(isk+x)}$$

$$= (g_1 h_2^{s_{uid}} h_1^{y})^{1/(isk+x)}.$$

Therefore A_{uid} is also distributed correctly.

For all other queries, s_{uid}, x_{uid} are chosen randomly and independently and so are distributed correctly. A_{uid} is also distributed correctly due to the following:

$$A_{uid} = B_l (B_l^{((x-x_l)\nu_1-1)/\nu_2} g_1^{\nu_1/\nu_2})^{s_{uid}+\theta y} = B_l (B_l^{((x-x_l)\nu_1-1)/\nu_2} B_l^{\nu_1(isk+x_l)/\nu_2})^{s_{uid}+\theta y}$$

ADDU(uid)

if $uid \in \mathsf{HUL} \cup \mathsf{CUL}$ **return** $\perp$ **else** $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}$ **return** accept

SNDI($uid, (H, \pi_H)$)

if $uid \in \mathsf{HUL}$ **return** $\perp$ **else** $\mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{uid\}, l \leftarrow l + 1$
Extract y as a witness of π_H
if $l = k$ $s_{uid} \leftarrow \nu_2 - \theta y, x_{uid} \leftarrow x, A_{uid} \leftarrow g_1^{\nu_1}, y_{uid} \leftarrow y$
else $x_{uid} \leftarrow x_l, s_{uid} \leftarrow_\$ \mathbb{Z}_p^*, A_{uid} \leftarrow B_l(B_l^{((x-x_l)\nu_1-1)/\nu_2} g_1^{\nu_1/\nu_2})^{s_{uid}+\theta y}, y_{uid} \leftarrow y$
return $((A_{uid}, x_{uid}, s_{uid}), \mathsf{cont})$

SIGN(uid, m)

if $uid \notin \mathsf{HUL}$ **return** $\perp$
if y_{uid} undefined $y_{uid} \leftarrow_\$ \mathbb{Z}_p^*$
$\alpha, \beta \leftarrow_\$ \mathbb{Z}_p^*, \mu_1 \leftarrow g^\alpha, \mu_2 \leftarrow cpk^\alpha h^{y_{uid}}, a \leftarrow_\$ \mathbb{Z}_p^*, A' \leftarrow g_1^a, \hat{A} \leftarrow \Gamma^a, d \leftarrow_\$ \mathbb{G}_1$
Simulate π using $A', \hat{A}, d, \mu_1, \mu_2, m$
return $((\mu_1, \mu_2), (A', \hat{A}, d, \pi))$

$\mathcal{A}'(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \tilde{g}_1, S_1, ..., S_q, g_2, w)$

$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_2$ given as input, $ipk \leftarrow w$, HUL, CUL $\leftarrow \emptyset$
g, h chosen as in CLS.Setup, csk, cpk chosen as in CLS.CKGen
$\forall i \in [1, q-1]$ $x_i \leftarrow_\$ \mathbb{Z}_p^*$
$\mu \leftarrow_\$ \mathbb{Z}_p^*$, let $f(X) = \prod_{i=1}^{q-1}(X + x_i) = \sum_{i=0}^{q-1} \zeta_i X^i, g_1 \leftarrow \prod_{i=0}^{q-1} S_i^{\mu\zeta_i}, \Gamma \leftarrow \prod_{i=0}^{q-1} S_{i+1}^{\mu\zeta_i}$
$x, \nu_1, \nu_2 \leftarrow_\$ \mathbb{Z}_p^*, h_2 \leftarrow ((\Gamma g_1^x)^{\nu_1} g_1^{-1})^{1/\nu_2}, \theta \leftarrow_\$ \mathbb{Z}_p^*, h_1 \leftarrow h_2^\theta, gpk \leftarrow (cpk, ipk, \mathsf{param})$
$\forall i \in [1, q-1]$ let $f_i(X) = \prod_{j=1, j\neq i}^{q-1}(X + x_i) = \sum_{j=0}^{q-2} \eta_{i,j} X^j, B_i \leftarrow \prod_{j=0}^{q-2} S_j^{\mu\eta_{i,j}}$
$l \leftarrow 0, k \leftarrow_\$ [1, q], ((m_1, \mu_1, \sigma_1), ..., (m_r, \mu_r, \sigma_r)) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SIGN}}(gpk, csk)$
Parse $\sigma_i = (A_i', \hat{A}_i, d_i, \pi_i)$
$\forall i \in [1, r]$ $\tilde{H}_i \leftarrow \mu_{i,2}\mu_{i,1}^{-csk}$
if $\exists i \in [1, r]$ s.t. $\forall uid \in \mathsf{CUL}$ $\tilde{H}_i \neq h^{y_{uid}}$
  Extract $\tilde{x}, \tilde{y}, \tilde{r}_2, \tilde{r}_3, \tilde{s}'$ from π_i
  if $\tilde{r_3} = 0, \tilde{A} \leftarrow 1, \tilde{s} \leftarrow \tilde{s'}$
  else $\tilde{A} \leftarrow A_i'^{\tilde{r_3}}, \tilde{s} \leftarrow \tilde{s'} + \tilde{r_2}\tilde{r_3}$
  $s^* \leftarrow \tilde{s} + \theta\tilde{y}$ **return** $((\tilde{A} g_1^{-\nu_1 s^*/\nu_2})^{\frac{\nu_2}{\nu_2 - s^* - \nu_1 s^*(\tilde{x}-x)}}, \tilde{x})$
else return $\perp$

Figure 4.14: $\mathcal{A}'$ which breaks the q-SDH assumption, using $\mathcal{A}$ which breaks the traceability of CLS–DDH with probability ϵ

$$= B_l B_l^{(s_{uid}+\theta y)(\nu_1(isk+x)-1)/\nu_2} = (g_1 g_1^{(s_{uid}+\theta y)(\nu_1(isk+x)-1)/\nu_2})^{1/(isk+x_l)}$$

$$= (g_1 h_2^{s_{uid}+\theta y})^{1/(isk+x_l)} = (g_1 h_2^{s_{uid}} h_1^{y})^{1/(isk+x_l)}.$$

Therefore, answers to SNDI queries are distributed identically to the traceability experiment.

Simulating the SIGN Oracle. For the SIGN oracle, μ_1, μ_2 are chosen identically to the traceability experiment. As a is chosen randomly and independently, A', d are chosen randomly, and as $\Gamma = g_1^{isk}$, $\hat{A} = A'^{isk}$, therefore $(A', \hat{A}, d)$ are distributed identically to CLS.Sign. Due to the zero-knowledge property of SPK, π can be simulated identically to the traceability experiment using $A', \hat{A}, d, \mu_1, \mu_2, m$. Therefore, answers to these queries are distributed identically to the traceability experiment.

Reduction to q-SDH. We assume $\mathcal{A}$ is successful. Then it outputs at least $q+1$ valid signatures that are unlinkable and not returned by the SIGN oracle for an uncorrupted user. As there are at least $q+1$ unlinkable signatures, there will be i such that $\forall uid \in \mathsf{CUL}, \quad \tilde{H}_i \neq h^{y_{uid}}$. As the signatures were not returned by the SIGN oracle, we can extract the witnesses for π_i due to soundness of the zero-knowledge proofs used.

We now show that we correctly extract a BBS+ signature $\tilde{A}$ on $\tilde{s}, \tilde{x}, \tilde{y}$ from π_H.

If $\tilde{r_3} = 0$, then $g_1 h_1^{\tilde{y}}, h_2^{\tilde{s'}} = 1$. Therefore $\tilde{A} = (g_1 h_1^{\tilde{y}} h_2^{\tilde{s}})^{1/(isk+\tilde{x})} = 1$.

If $\tilde{r_3} \neq 0$, then $\hat{A}_i = A'^{-\tilde{x}}_i d_i h_2^{\tilde{r_2}}$, and $\hat{A}_i = A'^{isk}_i$, therefore $A'^{isk+\tilde{x}}_i = d_i h_2^{\tilde{r_2}} = (g_1 h_1^{\tilde{y}} h_2^{\tilde{s'}})^{1/\tilde{r_3}} h_2^{\tilde{r_2}}$ $= (g_1 h_1^{\tilde{y}} h_2^{\tilde{s'}+\tilde{r}_2\tilde{r}_3})^{1/\tilde{r_3}} = (g_1 h_1^{\tilde{y}} h_2^{\tilde{s}})^{1/\tilde{r_3}}$. Therefore $\tilde{A} = A'^{\tilde{r_3}}_i = (g_1 h_1^{\tilde{y}} h_2^{\tilde{s}})^{1/(isk+\tilde{x})}$.

There are three possible cases for the BBS+ signature $\tilde{A}, \tilde{x}, \tilde{y}, \tilde{s}$.

Consider the first case, when $\tilde{x} \notin \{x_1, ..., x_{q-1}\} \cup \{x\}$. If $\nu_2 - s^* - \nu_1 s^*(\tilde{x} - x) = 0$, then $\nu_1 = \frac{\nu_2 - s^*}{s^*(\tilde{x}-x)}$. If $s^* = 0$, then $\nu_2 = 0$, which is not possible. Therefore the adversary can obtain ν_1 and so break the discrete logarithm problem, which is implied by the q-SDH problem.

We now show that $\mathcal{A}'$ outputs $g_1^{1/(isk+\tilde{x})}$, with which we can obtain a solution to the q-SDH

problem.

$$(\tilde{A}g_1^{-\nu_1 s^*/\nu_2})^{\frac{\nu_2}{\nu_2 - s^* - \nu_1 s^*(\tilde{x}-x)}} = (g_1 h_2^{s^*})^{\frac{\nu_2}{(isk+\tilde{x})(\nu_2 - s^* - \nu_1 s^*(\tilde{x}-x))}} g_1^{\frac{-\nu_1 s^*}{\nu_2 - s^* - \nu_1 s^*(\tilde{x}-x)}}$$

$$= (g_1 g_1^{s^*(\nu_1(isk+x)-1)/\nu_2})^{\frac{\nu_2}{(isk+\tilde{x})(\nu_2 - s^* - \nu_1 s^*(\tilde{x}-x))}} g_1^{\frac{-\nu_1 s^*}{\nu_2 - s^* - \nu_1 s^*(\tilde{x}-x)}}$$

$$= g_1^{\frac{s^*\nu_1(isk+x) - s^* + \nu_2 - s^*\nu_1(isk+\tilde{x})}{(isk+\tilde{x})(\nu_2 - s^* - \nu_1 s^*(\tilde{x}-x))}} = g_1^{1/(isk+\tilde{x})}.$$

Given this, which is a forgery as $\tilde{x} \notin \{x_1, ..., x_{q-1}\}$, $\mathcal{A}$ can break the q-SDH assumption as shown in [20].

For the second case, if for some uid, $\tilde{x} = x_{uid}$, and $\tilde{A} = A_{uid}$, but $\tilde{y} \neq y_{uid}$, then $\mathcal{A}'$ can break the discrete log assumption implied by the q-SDH assumption. This is because $log_{h_2}(h_1) = \frac{\tilde{s} - s_{uid}}{y_{uid} - \tilde{y}}$.

Finally we consider the third case, where $\tilde{x} \in \{x_1, ..., x_{q-1}\} \cup \{x\}$, but for $\tilde{x} = x_{uid}$, then $\tilde{A} \neq A_{uid}$. We assume $\tilde{x} = x$, which occurs with probability $1/q$. As $\tilde{A} \neq A_{uid}$, $\nu_2 - s^* = (s_{uid} - \tilde{s}) + \theta(y - \tilde{y}) \neq 0$.

We now show that $\mathcal{A}'$ outputs $g_1^{1/(isk+\tilde{x})}$, with which we can obtain a solution to the q-SDH problem:

$$(\tilde{A}g_1^{-\nu_1 s^*/\nu_2})^{\frac{\nu_2}{\nu_2 - s^*}} = (g_1 h_2^{s^*})^{\frac{\nu_2}{(isk+x)(\nu_2 - s^*)}} g_1^{\frac{-\nu_1 s^*}{\nu_2 - s^*}}$$

$$= (g_1 g_1^{s^*(\nu_1(isk+x)-1)/\nu_2})^{\frac{\nu_2}{(isk+x)(\nu_2 - s^*)}} g_1^{\frac{-\nu_1 s^*}{\nu_2 - s^*}} = g_1^{\frac{s^*\nu_1(isk+x) - s^* + \nu_2 - s^*\nu_1(isk+x)}{(isk+x)(\nu_2 - s^*)}} = g_1^{1/(isk+x)}.$$

Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20].

Therefore, if $\mathcal{A}$ is successful with probability ϵ, $\mathcal{A}'$ solves the q-SDH problem with probability at least ϵ/q.

Therefore, our CLS–DDH construction satisfies traceability.

□

4.5 Instantiation of SPK and Efficiency

We now discuss how to securely instantiate the online-extractable SPKs used in our CLS–DDH construction and state the computational cost and lengths of signatures and pseudonyms.

4.5.1 Instantiation of SPKs

We have two non-interactive zero–knowledge proofs of knowledge in our scheme:

$$\pi_H = \mathsf{SPK}\{(\underline{y}) : H = h_1^y\}(n)$$

used in the join protocol for proving knowledge of y in $H = h_1^y$, and

$$\pi = \mathsf{SPK}\{(x, \underline{y}, r_2, r_3, s', \alpha) : \mu_1 = g^\alpha \wedge \mu_2 = cpk^\alpha h^y$$

$$\wedge \hat{A}/d = A'^{-x} h_2^{r_2} \wedge g_1 h_1^y = d^{r_3} h_2^{-s'}\}(m)$$

proving knowledge of a BBS+ signature on y and that μ encrypts the same y. In both cases we need the witness y to be online extractable. For this, we additionally encrypt y under a public key that needs to be added to param (and to which in security proof we will know the secret key for), and extend π and π_H to prove that the additional encryption contains the same y that is used in the rest of the proof. For the verifiable encryption of y we use Paillier encryption [40], which is secure under the DCR assumption [109]. Proving correct encryption for such Paillier ciphertexts can be done using standard techniques from [40].

For transforming interactive into non-interactive zero–knowledge proofs we rely on the Fiat-Shamir heuristic that ensures security in the random oracle model. Due to this, we can now state Corollary 4.1.

Corollary 4.1. The CLS–DDH construction presented in Section 4.3.1, with the SPK instantiated as described, is a secure CLS as defined in Section 4.2 under the DDH, q-SDH and DCR assumption in the random oracle model.

4.5.2 Computational Cost

We provide the operations required for the entities involved in the scheme in Table 4.1. We denote k exponentiation in group $\mathbb{G}_i$ by $k\mathsf{exp}_{\mathbb{G}_i}$, k hash function calls by $k\mathsf{hash}$, and k pairing operations by $k\mathsf{pair}$. We denote k exponentiations in $\mathbb{Z}_{n^2}^*$ due to the Paillier encryption used, by $k\mathsf{exp}_{\mathbb{Z}_{n^2}^*}$.

Entity	Algorithm	Computational Cost
User	CLS.Sign	$16\mathsf{exp}_{\mathbb{G}_1} + 15\mathsf{exp}_{\mathbb{Z}_{n^2}} + 1\mathsf{hash}$
Verifier	CLS.Verify	$12\mathsf{exp}_{\mathbb{G}_1} + 11\mathsf{exp}_{\mathbb{Z}_{n^2}} + 1\mathsf{hash} + 2\mathsf{pair}$
	CLS.Blind	$6\mathsf{exp}_{\mathbb{G}_1}$
	CLS.Unblind	$2\mathsf{exp}_{\mathbb{G}_1}$
Converter	CLS.Convert (k pseudonyms input)	$7k\mathsf{exp}_{\mathbb{G}_1}$

Table 4.1: Computational costs for our CLS–DDH instantiation

4.5.3 Pseudonym and Signature Length

We provide the sizes of pseudonyms and signatures in terms of the number of group elements in Table 4.2. We denote the length required to represent k elements in $\mathbb{G}_1$ as $k\mathbb{G}_1$, k outputs of a hash function as kH, and k elements in $\mathbb{Z}_{n^2}^*$, due to the Paillier encryption used, as $k\mathbb{Z}_{n^2}^*$.

Pseudonym				Signature
Original μ	Blinded $c\mu$	Converted $\overline{c\mu}$	Unblinded and Converted $\overline{\mu}$	σ
$2\mathbb{G}_1$	$3\mathbb{G}_1$	$2\mathbb{G}_1$	$1\mathbb{G}_1$	$3\mathbb{G}_1\ 6\mathbb{Z}_p\ 1H\ 6\mathbb{Z}_{n^2}^*$

Table 4.2: Size of pseudonyms and signatures for our CLS–DDH instantiation

4.6 Summary

In this chapter we have proposed a new form of group signatures that support flexible and controlled linkability. Data can be collected in an authenticated and fully unlinkable form, whilst still allowing the data to be obliviously relinked by queries to a central entity. We have formalised the required security properties in a dynamic model and proposed

an efficient scheme that satisfies these requirement under discrete logarithm and Paillier assumptions in the random oracle model.

Chapter 5

Commuting Group Signatures

Contents

5.1 Introduction

This chapter introduces commuting group signatures, which allow group signatures to be blinded whilst remaining publicly verifiable. We present a formal security model for this new primitive and a construction that provably satisfies this model.

5.1.1 Motivation and Background

The *opener* in standard group signatures is the central entity that is invoked when a signature is de-anonymised. As discussed earlier, this role poses a privacy risk, as the

opener learns all messages and identities of users whose signatures are under dispute.

Commuting group signatures mitigate this by realising the opening obliviously. We model blinding by introducing another type of entity — the *data processors* $\mathcal{P}$. When requesting signatures to be opened, the signature and message get blinded under $\mathcal{P}$'s key and the opener returns a blinded user identity that only the data processor can unblind. The commuting aspect — following the notation of commuting signatures by Fuchsbauer [66]— is used to preserve the validity of a group signature during the various blinding operations.

A data processor may entrust a *data lake* to collect and store data for them. The data lake should not be able to see the messages and user identities, however they should still be able to check that they hold valid user data suitable for processing. Commuting group signatures allow a user to sign data and blind messages under the key of a trusted data processor. The data lake can still verify these blinded signatures, and pass them on to the data processor on request. The data processor can then unblind the messages/ signatures. If the data processor wishes to know the correlation of data by user, they can request that the opener obliviously opens these signatures.

Commuting group signatures can also be used as a building block in the context of group signatures with selective linkability. The fact that the blinded signatures can be verified, allows the converter to check inputs are well formed. This means we no longer need the honest–but–curious assumption for the verifier. This also allows authentication to be preserved, provided we still assume that the converter is honest–but–curious. This is because, the converter can attest that their inputs were valid blinded group signatures. Assuming that the converter is honest–but–curious this ensures that their output originates from valid user data and has been honestly blinded and converted.

5.1.2 Existing Work

Verifiably encrypted signatures [22] allow a signer to sign a message, encrypt the signature and prove that this is an encryption of a valid signature. In [66], *commuting signatures* were introduced that provide a verifiably encrypted signature on an encrypted message, starting from either an unencrypted message–signature tuple, or an encrypted message. Commuting group signatures similarly prove that there exists a valid group signature on an

encrypted message. Therefore, as in commuting signatures [66], data is publicly verifiable even when encrypted.

5.1.3 Our Contribution

We formally define the functionality and security requirements of commuting group signatures, adapting the standard notions of *anonymity*, *traceability* and *non-frameability* of group signatures to this blinded setting.

We then build a construction that provably satisfies these requirements. At a high level: a user's and issuer's key pair is each a signing and verification key of an automorphic signature scheme [65]. When joining the group, the issuer signs the user's verification key, yielding the user's membership credential. During signing, the user encrypts its verification key under the opener's public key, and proves knowledge of a valid automorphic signature on the message with respect to this verification key, as well as knowledge of a valid credential for this key.

In order to blind signatures for opening while preserving their verifiability, we use zero-knowledge proofs that are *controlled malleable* [45], which can be realised with *Groth-Sahai* proofs [77]. This allows the signature and message to be encrypted under a blinding key and the proof transformed accordingly. Because the malleability is controlled, this does not affect the unforgeability of the signatures.

Finally, we prove that our construction is secure based on the DDH assumption, and the security of the automorphic signatures and the controlled malleable NIZKs.

5.2 Chapter Preliminaries

We now present all building blocks and assumptions that are needed for our CGS construction. We use ElGamal encryption as a chosen–plaintext secure, re-randomisable and homomorphic encryption scheme, and standard proof protocols, as defined in Chapter 2. We also use automorphic signatures [65] and controlled malleable proof protocols.

5.2.1 Automorphic Signatures

An *automorphic signature* [65] over a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ is an EUF–cma secure signature whose verification keys are contained in the message space. Moreover, the messages and signatures consist of elements of $\mathbb{G}_1$ and $\mathbb{G}_2$, and the verification predicate is a conjunction of pairing-product equations over the verification key, the message and the signature. They consist of the following algorithms:

- $\mathsf{ASetup}(1^\tau)$ outputs the parameters for the signature scheme $\mathsf{param}_{\mathsf{auto}}$.
- $\mathsf{AKeyGen}(\mathsf{param}_{\mathsf{auto}})$ on input the parameters, outputs a public verification key and a secret signing key, (apk, ask).
- $\mathsf{ASign}(ask, m)$ on input a signing key and message, outputs a signature Ω.
- $\mathsf{AVerify}(\Omega, apk, m)$: checks that Ω is a valid signature on m under apk.

We now present the automorphic signature scheme [65] that will be used in our concrete instantiation. We have rewritten the scheme so that the message is an element of $\mathbb{G}_1$ and the verification key is an element of $\mathbb{G}_2$. This can be done analogously so that the message is an element of $\mathbb{G}_2$ and the verification key is an element of $\mathbb{G}_1$.

$\mathsf{ASetup}(1^\tau)$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$\, \mathcal{G}(1^\tau), F, K, T \leftarrow\!\!\$\, \mathbb{G}_1 \quad \textbf{return } ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), F, K, T)$

$\mathsf{AKeyGen}(\mathsf{param}_{\mathsf{auto}})$

$ask \leftarrow\!\!\$\, \mathbb{Z}_p^*, apk \leftarrow g_2^{ask} \quad \textbf{return } (apk, ask)$

$\mathsf{ASign}(ask, M)$

$r, c \leftarrow\!\!\$\, \mathbb{Z}_p^*, A \leftarrow (K \cdot T^r \cdot M)^{1/(ask+c)}, C \leftarrow F^c, D \leftarrow g_2^c, R \leftarrow g_1^r, S \leftarrow g_2^r \quad \textbf{return } (A, C, D, R, S)$

$\mathsf{AVerify}(\Omega, apk, M)$

Check $e(A, apk \cdot D) = e(K \cdot M, g_2)e(T, S), e(C, g_2) = e(F, D), e(R, g_2) = e(g_1, S)$

5.2.2 Controlled Malleable NIZKs

A *controlled malleable proof* [45] for a relation $\mathcal{R}$ and *transformation class* $\mathcal{T}$, which gives the set of allowed transformations, consists of three algorithms constituting a regular non-interactive proof:

- $\mathsf{CRSSetup}(1^\tau)$: generates a common reference string σ_{crs}.
- $\mathcal{P}(\sigma_{\mathsf{crs}}, x, w)$: takes as input the common reference string σ_{crs}, as a well as a statement x and a witness such that $(x, w) \in \mathcal{R}$, and outputs a proof π.
- $\mathcal{V}(\sigma_{\mathsf{crs}}, x, \pi)$: takes as input the common reference string σ_{crs}, as a well as a statement x, and a proof π and outputs 1 if π is valid, and 0 otherwise.

As detailed in Chapter 2, such a proof is called *zero–knowledge* (NIZK) if there exists a PPT simulator (S_1, S_2) such that an adversary can't distinguish between proofs formed by the prover and proofs formed by the simulator; and a proof of knowledge (NIZKPoK) if there exists a PPT extractor (E_1, E_2) that can produce a valid witness from any valid proof. The fourth algorithm, specific to malleable proof systems, is:

- $\mathsf{ZKEval}(\sigma_{\mathsf{crs}}, T, x, \pi)$: which, on input σ_{crs}, a transformation $T = (T_{\mathsf{inst}}, T_{\mathsf{wit}})$ (in some transformation class $\mathcal{T}$), an instance x, and a proof π, outputs a mauled proof π' for instance $T_{\mathsf{inst}}(x)$.

The *controlled-malleable simulation-sound extractability* requirement reconciles malleability with simulation-sound extractability [52, 75] and requires that, for any instance x, if an adversary can produce a valid proof π that $x \in \mathcal{R}$ then an extractor can extract from π either a witness w such that $(x, w) \in \mathcal{R}$ or a previously proved instance x' and transformation $T \in \mathcal{T}$ such that $x = T_{\mathsf{inst}}(x')$. This guarantees that any proof that the adversary produces is either generated from scratch using a valid witness, or formed by applying a transformation from the class $\mathcal{T}$ to an existing proof.

Definition 5.1. Let $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ be a NIZKPoK system for an efficient relation R, with a simulator (S_1, S_2) and an extractor (SE_1, E_2). Let $\mathcal{T}$ be an allowable set of unary transformations for the relation $\mathcal{R}$ such that membership in $\mathcal{T}$ is efficiently testable.

Let SE_1 be an algorithm that, on input (1^τ), outputs $(\sigma_{\mathsf{crs}}, \tau_s, \tau_e)$ such that $(\sigma_{\mathsf{crs}}, \tau_s)$ is distributed identically to the output of S_1. Consider the following game with adversary $\mathcal{A}$:

- Step 1. $(\sigma_{\mathsf{crs}}, \tau_s, \tau_e) \leftarrow\!\!\$\ SE_1(1^\tau)$
- Step 2. $(x, \pi) \leftarrow\!\!\$\ \mathcal{A}^{S_2(\sigma_{\mathsf{crs}}, \tau_s, \cdot)}(\sigma_{\mathsf{crs}}, \tau_e)$
- Step 3. $(w, x', T) \leftarrow E_2(\sigma_{\mathsf{crs}}, \tau_e, x, \pi)$.

The proof system satisfies controlled-malleable simulation-sound extractability (CM-SSE) with respect to $\mathcal{T}$ if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that the probability (over the choices of SE_1, $\mathcal{A}$, and S_2) that $\mathcal{V}(\sigma_{\mathsf{crs}}, x, \pi) = 1$ and $(x, \pi) \notin Q$ (where Q is the set of queried statements and their responses) but either:

- $w \neq \perp$ and $(x, w) \notin \mathcal{R}$;
- $(x', T) \neq (\perp, \perp)$ and either $x' \notin Q_{\mathsf{inst}}$ (the set of queried instances), $x \neq T_{\mathsf{inst}}(x')$, or $T \notin \mathcal{T}$;
- $(w, x', T) = (\perp, \perp, \perp)$;

is at most $\mathsf{negl}(\tau)$.

In [45] *strong derivation privacy* for such proofs is also defined. This ensures simulated proofs are indistinguishable from those formed via a transformation.

Definition 5.2. For a malleable NIZK $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, \mathsf{ZKEval})$ with an associated simulator (S_1, S_2), a given adversary $\mathcal{A}$, and bit b, let $p_b^{\mathcal{A}}(\tau)$ be the probability of the event that $b' = 1$ in the following game:

- Step 1. $(\sigma_{\mathsf{crs}}, \tau_s) \leftarrow\!\!\$\ S_1(1^\tau)$;
- Step 2. $(\mathsf{state}, x_1, \pi_1, ..., x_q, \pi_q, T) \leftarrow\!\!\$\ \mathcal{A}(\sigma_{\mathsf{crs}}, \tau_s)$;

- Step 3. If $\mathcal{V}(\sigma_{\mathsf{crs}}, x_i, \pi_i) = 0$ for some i, $(x_1, \cdots, x_q)$ is not in the domain of T_{inst}, or $T \notin \mathcal{T}$, abort and output $\perp$. Otherwise, form

$$\pi \leftarrow_{\$} \begin{cases} S_2(\sigma_{\mathsf{crs}}, \tau_s, T_{\mathsf{inst}}(x_1, \cdots, x_q)), & \text{if } b = 0 \\ \mathsf{ZKEval}(\sigma_{\mathsf{crs}}, T, \{x_i, \pi_i\}_{i \in [q]}), & \text{if } b = 1 \end{cases}$$

- Step 4. $b' \leftarrow_{\$} \mathcal{A}(\mathsf{state}, \pi)$.

The proof system is *strongly derivation private* if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that $|p_0^{\mathcal{A}}(\tau) - p_1^{\mathcal{A}}(k)| < \mathsf{negl}(\tau)$.

Putting these two definitions together, a cm-NIZK is defined to be a proof system that is CM-SSE, strongly derivation private, and zero–knowledge.

A cm-NIZK Construction We now present the construction given in [45] that provably satisfies the above definitions. Let $\mathcal{R}$ be an efficient relation, and suppose $\mathcal{T}$ is an allowable set of transformations for $\mathcal{R}$ that contains the identity transformation; suppose further that membership in $\mathcal{T}$ is efficiently testable.

Let $(\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{Verify})$ be a structure preserving signature scheme that is EUF-cma secure as defined in Sections 2.5.3 and 2.5.4. Let $(\mathsf{CRSSetup}_{\mathsf{WI}}, \mathcal{P}_{\mathsf{WI}}, \mathcal{V}_{\mathsf{WI}}, \mathsf{ZKEval}_{\mathsf{WI}})$ be a non-interactive witness indistinguishable proof of knowledge (NIWIPoK), as defined in Section 2.6.1, for the relation: $((x, vk), (w, x', T, \Omega)) \in \mathcal{R}_{\mathsf{WI}}$ if $(x, w) \in \mathcal{R}$ or $\mathsf{Verify}(vk, \Omega, x') = 1$, $x = T_{\mathsf{inst}}(x')$, and $T \in \mathcal{T}$. We also require the NIWIPoK to be derivation private with respect to $\mathcal{R}_{\mathsf{WI}}$ and a set of transformations $\mathcal{T}_{\mathsf{WI}}$ such that for every $T' = (T'_{\mathsf{inst}}, T'_{\mathsf{wit}}) \in \mathcal{T}$ there exists a $T_{\mathsf{WI}}(T') \in \mathcal{T}_{\mathsf{WI}}$. For $T_{\mathsf{WI}}(T') = (T_{\mathsf{inst,WI}}, T_{\mathsf{wit,WI}})$ we require that $T_{\mathsf{inst,WI}}(x, vk) = (T'_{\mathsf{inst}}(x), vk)$, and $T_{\mathsf{wit,WI}}(w, x', T, \Omega) = (T'_{\mathsf{wit}}(w), x', T' \circ T, \Omega)$.

The cm-NIZK is defined as follows:

- $\mathsf{CRSSetup}(1^\tau)$: Generate $\sigma_{\mathsf{crs,WI}} \leftarrow_{\$} \mathsf{CRSSetup}_{\mathsf{WI}}(1^\tau)$ and $(vk, sk) \leftarrow_{\$} \mathsf{SigKGen}(1^\tau)$, output $\sigma_{\mathsf{crs}} = (\sigma_{\mathsf{crs,WI}}, vk)$.
- $\mathcal{P}(\sigma_{\mathsf{crs}}, x, w)$: Output $\pi \leftarrow_{\$} \mathcal{P}_{\mathsf{WI}}(\sigma_{\mathsf{crs,WI}}, x_{\mathsf{WI}}, w_{\mathsf{WI}})$, where $x_{\mathsf{WI}} = (x, vk)$ and $w_{\mathsf{WI}} = (w, \perp, \perp, \perp)$.

- $\mathcal{V}(\sigma_{\mathsf{crs}}, x, \pi)$: Output $\mathcal{V}_{\mathsf{WI}}(\sigma_{\mathsf{crs},\mathsf{WI}}, x_{\mathsf{WI}}, \pi)$ where $x_{\mathsf{WI}} = (x, vk)$.
- $\mathsf{ZKEval}(\sigma_{\mathsf{crs}}, T, x, \pi)$: Output $\mathsf{ZKEval}_{\mathsf{WI}}(\sigma_{\mathsf{crs},\mathsf{WI}}, T_{\mathsf{WI}}(T), x_{\mathsf{WI}}, \pi)$ where $x_{\mathsf{WI}} = (x, vk)$.

Instantiation In [45] it is shown that Groth Sahai proofs [77] can be used to instantiate the derivation private NIWIPoKs used in this construction, and a structure preserving signature was identified [44].

5.3 Definition and Security Model for Commuting Group Signatures

In this section we first introduce the syntax and generic functionality of commuting group signatures CGS and then present the desirable security and privacy properties for such schemes.

The entities involved in a commuting group signature scheme are an *issuer* $\mathcal{I}$, a set of users $\mathcal{U} = \{uid_i\}$, an *opener* $\mathcal{O}$, and — additionally — *data processors* $\mathcal{P}$. Once joining the group via the issuer $\mathcal{I}$, a user can either sign as in standard group signatures, or output blinded signatures directly (with respect to the blinding public key of $\mathcal{P}$). Standard signatures and attached messages can also be blinded afterwards. Opening can be done for both standard group signatures and blinded ones, where it returns the user identity in plain and in blinded form respectively. Signatures, messages, or user identities that are blinded can only be retrieved by the designated data processor $\mathcal{P}$. Our model allows for several data processors. However, the blinding public key must be input to CGS.Sign. This enables users, and only them, to choose their preferred and trusted data processor, but also limits the flexibility as they have to be chosen at the moment the signatures are created.

5.3.1 Syntax of CGS

Definition 5.3 (CGS)**.** A commuting group signature CGS scheme consists of the following algorithms:

Setup and Key Generation. We refer to $(\mathsf{param}, ipk, opk)$ as the group public key gpk, and $\mathcal{BK}$ as the public/ private key space induced by CGS.BKGen.

CGS.Setup$(1^\tau) \rightarrow$ **param** on input the security parameter 1^τ, outputs param, the public parameters for the scheme.

CGS.IKGen(**param**) $\rightarrow (ipk, isk)$ performed by the issuer $\mathcal{I}$; outputs the issuer secret key isk, and the issuing public key ipk.

CGS.OKGen(**param**) $\rightarrow (opk, osk)$ performed by the opener $\mathcal{O}$; outputs the opener secret key osk, and the opener public key opk.

CGS.BKGen(**param**) $\rightarrow (bpk, bsk)$ performed by a data processor $\mathcal{P}$; outputs a blinding secret key, bsk, and blinding public key, bpk.

Join, Sign and Verify. As in dynamic group signatures, a user joins via the issuer, and they can then sign standard and blinded signatures. In contrast to the model for standard group signatures [14], we split the signature into a pseudonym μ and a signature σ, for ease of notation. All group signatures following the predominant Sign-Encrypt-Prove paradigm can be matched to this notation.

CGS.UKGen(**param**) $\rightarrow (\mathbf{usk}[uid], \mathbf{upk}[uid])$ a user generates their own secret key $\mathbf{usk}[uid]$ and their user public key $\mathbf{upk}[uid]$.

$\langle$**CGS.Join**$(gpk, \mathbf{usk}[uid], \mathbf{upk}[uid])$, **CGS.Issue**$(isk, gpk, \mathbf{upk}[uid])\rangle$ a user uid joins the group by engaging in an interactive protocol with the issuer $\mathcal{I}$. The user uid and issuer $\mathcal{I}$ perform algorithms CGS.Join and CGS.Issue respectively. These are input a state and an incoming message respectively, and output an updated state, an outgoing message, and a decision, either cont, accept, or reject. The initial input to CGS.Join is the group public key, gpk, their user secret key $\mathbf{usk}[uid]$, and user public key $\mathbf{upk}[uid]$, whereas the initial input to CGS.Issue is the issuer secret key, isk, the issuer public key ipk and user public key $\mathbf{upk}[uid]$. If the user uid accepts, CGS.Join has a private output of $\mathbf{gsk}[uid]$.

CGS.Sign$(gpk, bpk, \mathbf{gsk}[uid], m) \rightarrow (\mu, \sigma)$ performed by the user with identifier uid, with input the group public key gpk, blinding public key bpk, the user's secret key $\mathbf{gsk}[uid]$, and a message m; outputs a pseudonym μ and signature σ.

CGS.Verify$(gpk, bpk, m, \mu, \sigma) \rightarrow \{0, 1\}$ performed by a verifier; outputs 1 if σ is a valid signature on m for pseudonym μ under the group public key gpk and blinding public key bpk, and 0 otherwise.

Blinding. A successfully joined user can also create blinded group signatures directly, where blinding is done with respect to bpk. Anyone knowing a standard group signature can also blind it for the particular data processor. Importantly, despite being fully blinded, the correctness of these signatures can still be verified.

CGS.BlindSign$(gpk, bpk, \mathbf{gsk}[uid], m) \rightarrow (c, c\mu, c\sigma)$ performed by the user with identifier uid with input the group public key gpk, including the blinding public key bpk, the user's secret key $\mathbf{gsk}[uid]$, and a message m; outputs a blinded message, pseudonym, and signature.

CGS.BlindUser$(bpk, upk) \rightarrow cupk$ performed on input a user public key upk, and the blinding public key bpk; outputs a blinded user public key.

CGS.Blind$(gpk, bpk, m, \mu, \sigma) \rightarrow (c, c\mu, c\sigma)$ performed by the verifiers with input the group public key gpk including the blinding public key bpk, a message m, a pseudonym μ and a signature σ; outputs a blinded message, pseudonym, and signature.

CGS.BlindVerify$(gpk, bpk, c, c\mu, c\sigma) \rightarrow \{0, 1\}$ performed by a verifier; outputs 1 if $c\sigma$ is a valid blinded signature on c for pseudonym $c\mu$ under the group public key gpk, and 0 otherwise.

Opening and Unblinding. Both standard and blinded group signatures can be de-anonymised by the opener $\mathcal{O}$, which outputs a plain user public key and a blinded one respectively. The unblinding of user public keys or messages can be done with the corresponding bsk only.

CGS.Open$(gpk, bpk, osk, m, \mu, \sigma) \rightarrow \{upk, \perp\}$ performed by the opener $\mathcal{O}$, on input their opening secret key, a message m, pseudonym μ and signature σ; outputs a user public key or a failure symbol $\perp$.

CGS.OpenBlind$(gpk, bpk, osk, c, c\mu, c\sigma) \rightarrow \{cupk, \perp\}$ performed by the opener $\mathcal{O}$, on input their opening secret key, and a blinded message c, pseudonym $c\mu$ and signature $c\sigma$; outputs a blinded user public key or a failure symbol $\perp$.

CGS.UnblindUser$(bsk, cupk) \rightarrow upk$ on input a blinded user public key $cupk$, and the blinding secret key bsk, outputs a user public key.

CGS.UnblindM$(bsk, c) \rightarrow m$ on input a blinded message c, and the blinding secret key bsk; outputs the message m.

Re–randomisation. When used in privacy-preserving protocols, such as the CLS+ scheme presented in Chapter 6, it may be useful if both types of signatures can be *re-randomised*, which we therefore make explicit here. When re-randomising blinded signatures, the algorithm also refreshes the blinded message. Sometimes we do not need the full tuple $c, c\mu, c\sigma$ to be re-randomised and omit the unnecessary outputs for brevity.

CGS.RRand$(gpk, bpk, m, \mu, \sigma) \rightarrow (\mu', \sigma')$ performed by a verifier, on input the group public key, the blinding public key, and a message m, pseudonym μ and signature σ; outputs a re-randomised pseudonym and signature.

CGS.RRandBlind$(gpk, bpk, c, c\mu, c\sigma) \rightarrow (c', c\mu', c\sigma')$ performed by a verifier, on input the group public key, the blinding public key, and a blinded message c, pseudonym $c\mu$ and signature $c\sigma$; outputs a re-randomised blinded message, pseudonym and signature.

5.3.2 Security Properties of CGS

Like standard dynamic group signatures [14], commuting group signatures must satisfy the *anonymity*, *traceability* and *non–frameability* requirements. However, the anonymity requirement must now take the blinding capabilities into account, and capture the opening of blinded signatures and its impact in particular. Likewise, as blinded signatures can still be verified against the group public key, the traceability and non-frameability requirements must be extended as well. Finally, we additionally require that blinded signatures cannot be linked to the signature before blinding, which we formalise in the *blindness* definition.

Oracles and State. Our security notions make use a number of oracles: ADDU, SNDU, SNDI, SIGN, OPEN, BOPEN, which keep joint state.

We present the detailed description of these oracles in Figure 5.1 and an overview of them and their maintained records. We do not provide oracles for all of the CGS algorithms, however we require that they are indistinguishable from algorithms we do capture in our security model.

ADDU (join of honest user and honest issuer) Creates a new honest user for *uid* and internally runs a join protocol between the honest user and honest issuer. At the end, the honest user's secret key $\mathbf{gsk}[uid]$ and public key $\mathbf{upk}[uid]$ are generated, and from then on signing/ opening queries for *uid* will be allowed.

SNDU (join of honest user and corrupt issuer) Creates a new honest user for *uid* and runs the join protocol on behalf of *uid* with the corrupt issuer. If the join session completes, the oracle will store the user's secret key $\mathbf{gsk}[uid]$, and public key $\mathbf{upk}[uid]$.

SNDI (join of corrupt user and honest issuer) Runs the join protocol on behalf of the honest issuer with corrupt users. If the issue session completes, $\mathbf{upk}[uid]$ will be stored. For joins of honest users, the ADDU oracle must be used.

SIGN This oracle returns signatures for honest users that have successfully joined (via ADDU or SNDU, depending on the game).

OPEN This oracle opens standard signatures, and outputs a user public key.

BOPEN This oracle opens blinded signatures, and outputs a blinded user public key.

All oracles have access to the following records maintained as global state:

HUL List of *uid*s of honest users, initially set to $\emptyset$. New honest users can be added by queries to the ADDU oracle (when the issuer is honest) or SNDU oracle (when the issuer is corrupt).

CUL List of corrupt users that have requested to join the group, initially set to $\emptyset$. New corrupt users can be added through the SNDI oracle if the issuer is honest. If the issuer is corrupt, we do not keep track of corrupt users.

ADDU(uid)

if $uid \in \mathsf{HUL} \cup \mathsf{CUL}$ **return** $\perp$
$(\mathbf{usk}[uid], \mathbf{upk}[uid]) \leftarrow\!\!\$\ \mathsf{CGS.UKGen}(\mathsf{param})$
$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp$
$\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, \mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk, \mathbf{usk}[uid], \mathbf{upk}[uid])$
$\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk, \mathbf{upk}[uid])$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CGS.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, \perp)$
while $\mathbf{dec}^{uid} = \mathsf{cont}$
  $(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Join}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CGS.Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Issue}})$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CGS.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Join}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$
  $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$,
return $(\mathsf{accept}, \mathbf{upk}[uid])$

SNDI($uid, M_{\mathsf{in}}, upk$)

if $uid \in \mathsf{HUL}$ **return** $\perp$
if $uid \notin \mathsf{CUL}$
  $\mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{uid\}, \mathbf{upk}[uid] \leftarrow upk$
  $\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, \mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk, upk)$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
$(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CGS.Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{in}})$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

SIGN(uid, m, bpk)

if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** $\perp$
$(\mu, \sigma) \leftarrow\!\!\$\ \mathsf{CGS.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)$
$\mathsf{SL} \leftarrow \mathsf{SL} \cup \{(uid, m, bpk)\}$
return (σ, μ)

SNDU(uid, M_{in})

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}$
  $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
  $(\mathbf{usk}[uid], \mathbf{upk}[uid]) \leftarrow\!\!\$\ \mathsf{CGS.UKGen}(\mathsf{param})$
  return $(\mathbf{upk}[uid], \mathsf{cont})$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if $\mathsf{st}^{uid}_{\mathsf{Join}}$ undefined $\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk, \mathbf{usk}[uid], \mathbf{upk}[uid])$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CGS.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

OPEN(m, μ, σ, bpk)

$upk \leftarrow \mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma)$
if $m = m^*$ and $upk = \mathbf{upk}[uid^*_d]$ for $d \in \{0, 1\}$
  return $\perp$
else return upk

BOPEN($c, c\mu, c\sigma, bpk, bsk$)

if $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
$\overline{upk} \leftarrow \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma)$
$m \leftarrow \mathsf{CGS.UnblindM}(bsk, c), upk \leftarrow \mathsf{CGS.UnblindUser}(bsk, \overline{upk})$
if $m = m^*$ and $upk = \mathbf{upk}[uid^*_d]$ for $d \in \{0, 1\}$
  return $\perp$
else return upk

Figure 5.1: Oracles used in our CGS security model

SL List of (uid, m, bpk) tuples requested from the SIGN oracle.

Correctness. The standard signature scheme should be correct as in the original definition for dynamic group signatures [14]. We further need to ensure that honestly generated *blinded* signatures verify and open correctly, and that the blinding can be reversed.

The game defining the correctness requirements for commuting group signatures is given in Figure 5.2.

Definition 5.4 (Correctness)**.** A commuting group signature scheme CGS satisfies correctness if for all adversaries $\mathcal{A}$: $\Pr[\mathbf{Exp}^{corr}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1] = 0$.

Commutative Behaviour of CGS. Our model now contains a multitude of algorithms, essentially providing all functionality for blinded as well as for unblinded inputs. To avoid a complexity blow-up in the security games, we require the indistinguishability of certain

Experiment: $\mathsf{Exp}^{\mathrm{corr}}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CGS.Setup}(1^\tau), (isk, ipk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (osk, opk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$
$(bsk, bpk) \leftarrow_\$ \mathsf{CGS.BKGen}(\mathsf{param}), \mathsf{CUL}, \mathsf{HUL} \leftarrow \emptyset, gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(uid, m) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU}}(gpk)$, **if** $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** 0
$(\mu, \sigma) \leftarrow_\$ \mathsf{CGS.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)$, **if** $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) = 0$ **return** 1
$upk \leftarrow \mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma)$, **if** $\mathbf{upk}[uid] \neq upk$ **return** 1
$(c, c\mu, c\sigma) \leftarrow_\$ \mathsf{CGS.Blind}(gpk, bpkm, \mu, \sigma)$
if $\mathsf{CGS.BlindVerify}(gpk, bpk, c, c\mu, c\sigma) = 0$ **return** 1
if $\mathsf{CGS.UnblindM}(bsk, c) \neq m$ **return** 1
$upk' \leftarrow \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma))$
if $upk \neq upk'$ **return** 1

Figure 5.2: Game defining correctness for CGS

operations, or in fact their *commutative behaviour*.

First, we require that directly blinded signatures are indistinguishable from standard ones that were blinded later:

$$\mathsf{CGS.Blind}(gpk, bpk, m, (\mathsf{CGS.Sign}(gpk, bpk, \mathbf{gsk}[uid], m))) \approx$$

$$\mathsf{CGS.BlindSign}(gpk, bpk, \mathbf{gsk}[uid], m).$$

Likewise, outputs of CGS.BlindUser must be indistinguishable from the output of CGS.OpenBlind on a valid blinded signature with the same public key:

$$\mathsf{CGS.BlindUser}(bpk, (\mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma))) \approx$$

$$\mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma).$$

Further, we want that verification and opening of blinded signatures $(c, c\mu, c\sigma) \leftarrow_\$ \mathsf{CGS.Blind}(gpk, bpk, m, \mu, \sigma)$ must yield the same result as their unblind counterparts for $(\mu, \sigma) \leftarrow_\$ \mathsf{CGS.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)$:

$$\mathsf{CGS.BlindVerify}(gpk, bpk, c, c\mu, c\sigma) = \mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma)$$

$$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma)) =$$

$$\mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma).$$

The full requirements for the commutative behaviour of CGS are given in Figure 5.3. They

Experiment: $\mathsf{Exp}^{\text{indbsign-b}}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CGS.Setup}(1^\tau), (isk, ipk) \leftarrow\!\!\$\ \mathsf{CGS.IKGen}(\mathsf{param}), (osk, opk) \leftarrow\!\!\$\ \mathsf{CGS.OKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(uid, m, \mu, \sigma, bpk) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{ADDU}}(gpk, isk, osk)$
if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ or $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) = 0$ **return** 0
$(c_0, c\mu_0, c\sigma_0) \leftarrow\!\!\$\ \mathsf{CGS.Blind}(gpk, bpk, m, \mu, \sigma), (c'_1, c\mu'_1, c\sigma'_1) \leftarrow\!\!\$\ \mathsf{CGS.BlindSign}(gpk, bpk, \mathbf{gsk}[uid], m)$
$b^* \leftarrow\!\!\$\ \mathcal{A}(c_b, c\mu_b, c\sigma_b)$, **return** b^*

Experiment: $\mathsf{Exp}^{\text{indbuser-b}}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CGS.Setup}(1^\tau), (isk, ipk) \leftarrow\!\!\$\ \mathsf{CGS.IKGen}(\mathsf{param}), (osk, opk) \leftarrow\!\!\$\ \mathsf{CGS.OKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(m, \mu, \sigma, bpk) \leftarrow\!\!\$\ \mathcal{A}(gpk, isk, osk)$, **if** $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) = 0$ **return** 0
$cupk_0 \leftarrow \mathsf{CGS.OpenBlind}(gpk, bpk, osk, \mathsf{CGS.Blind}(gpk, bpk, m, \mu, \sigma))$
$cupk_1 \leftarrow\!\!\$\ \mathsf{CGS.BlindUser}(bpk, \mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma))$
$b^* \leftarrow\!\!\$\ \mathcal{A}(cupk_b)$, **return** b^*

Experiment: $\mathsf{Exp}^{\text{indbverif}}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CGS.Setup}(1^\tau), (isk, ipk) \leftarrow\!\!\$\ \mathsf{CGS.IKGen}(\mathsf{param}), (osk, opk) \leftarrow\!\!\$\ \mathsf{CGS.OKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(m, \mu, \sigma, bpk) \leftarrow\!\!\$\ \mathcal{A}(gpk, isk, osk), (c, c\mu, c\sigma) \leftarrow\!\!\$\ \mathsf{CGS.Blind}(gpk, bpk, m, \mu, \sigma)$
if $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) \neq \mathsf{CGS.BlindVerify}(gpk, bsk, \mathsf{CGS.Blind}(gpk, bpk, m, \mu, \sigma))$ **return** 1
if $\mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma) \neq \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma))$
return 1
else return 0

Figure 5.3: Games defining commutative behaviour of CGS

allow us to omit redundant oracles and simplify our model.

Definition 5.5 (Commutative Behaviour of CGS)**.** A commuting group signature scheme CGS satisfies commutative behaviour of CGS if for all ppt adversaries $\mathcal{A}$ the following advantages are negligible in τ: $|\Pr[\mathbf{Exp}^{indbsign-0}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1] - \Pr[\mathbf{Exp}^{indbsign-1}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]|$, $|\Pr[\mathbf{Exp}^{indbuser-0}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1] - \Pr[\mathbf{Exp}^{indbuser-1}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]|$, and $\Pr[\mathbf{Exp}^{indbverif}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]$.

Anonymity. This requirement ensures that group signatures cannot be de-anonymised by an adversary that has corrupted the issuer and data processor, but not the opener. The adversary has access to the opener, which they can query now on standard signatures as well as on blinded ones (via the BOPEN oracle). Opening blinded inputs might seem a risky choice, as it might lead to trivial wins by blinding and re-randomising the challenge signature. To avoid this, we opt for an RCCA-version of the classic (CCA) anonymity notion, and let the BOPEN oracle check whether it was queried on a blinded signature that opens to either of the challenge users and the challenge message, and abort in such

cases. In order to do so, the adversary must also provide the blinding key pair to the oracle. The key pair is first checked for correctness and then used to unblind the blinded message and user identity.

The indistinguishability requirement of $\mathsf{CGS.BlindSign} \approx \mathsf{CGS.Blind}(\mathsf{CGS.Sign})$ ensures that this requirement also covers the anonymity of signatures originated from CGS.BlindSign.

Definition 5.6 (CGS Anonymity)**.** A commuting group signature scheme CGS satisfies anonymity if for all polynomial–time adversaries $\mathcal{A}$, the following advantage is negligible in τ:

$$\left|\Pr[\mathsf{Exp}^{anon-1}_{\mathcal{A},\mathsf{CGS}}(\tau)=1]-\Pr[\mathsf{Exp}^{anon-0}_{\mathcal{A},\mathsf{CGS}}(\tau)=1]\right|.$$

Experiment: $\mathsf{Exp}^{\text{anon-b}}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CGS.Setup}(1^\tau), (isk, ipk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (osk, opk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(\mathsf{st}, uid_0^*, uid_1^*, m^*, bpk) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU,SIGN,OPEN,BOPEN}}(\mathsf{choose}, gpk, isk)$

if $uid_0^*, uid_1^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0^*], \mathbf{gsk}[uid_1^*] = \perp$ **return** 0

$(\mu^*, \sigma^*) \leftarrow_\$ \mathsf{CGS.Sign}(gpk, bpk, \mathbf{gsk}[uid_b^*], m^*)$

$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU,SIGN,OPEN,BOPEN}}(\mathsf{guess}, \mathsf{st}, \mu^*, \sigma^*)$ **return** b^*

Blindness. This requirement ensures that blinded group signatures and messages cannot be linked to the group signature and message they originate from, when the issuer and opener are corrupted. We also capture that CGS.BlindUser securely blinds user public keys. As the adversary has corrupted the issuer and opener, and this property does not distinguish between honest and corrupted users, we do not provide them with any oracle access.

Definition 5.7 (CGS Blindness)**.** A commuting group signature scheme CGS satisfies blindness if for all polynomial–time adversaries $\mathcal{A}$ the following advantage is negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{blind-0}_{\mathcal{A},\mathsf{CGS}}(\tau)=1]-\Pr[\mathbf{Exp}^{blind-1}_{\mathcal{A},\mathsf{CGS}}(\tau)=1]\right|.$$

Experiment: $\mathbf{Exp}^{blind-b}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CGS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$(bpk, bsk) \leftarrow_\$ \mathsf{CGS.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(\mathsf{st}, (upk_0, \mu_0, \sigma_0, m_0), (upk_1, \mu_1, \sigma_1, m_1)) \leftarrow_\$ \mathcal{A}(\mathsf{choose}, gpk, bpk, isk, osk)$

if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0$ **return** 0

$(c^*, c\mu^*, \sigma^*) \leftarrow_\$ \mathsf{CGS.Blind}(gpk, bpk, m_b, \mu_b, \sigma_b)), cupk \leftarrow_\$ \mathsf{CGS.BlindUser}(bpk, upk_b)$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, cupk, c^*, c\mu^*, \sigma^*)$ **return** b^*

Traceability. This requirement ensures that group signatures can always be traced to a user identifier from a join session. Here the adversary can corrupt the opener and data processor, but not the issuer. We formulate this property for blinded signatures only, but the guarantees carry over to standard ones as well, due to the commutative behaviour of

$$\mathsf{CGS.BlindVerify}(\mathsf{CGS.Blind}) = \mathsf{CGS.Verify} \text{ and}$$

$$\mathsf{CGS.UnblindUser}(\mathsf{CGS.OpenBlind}) = \mathsf{CGS.Open}.$$

That is, if an adversary could forge a standard group signature that does not trace to a join session, then it could blind this signature and so forge a valid *blinded* group signature (as required by the experiment).

Definition 5.8 (CGS Traceability)**.** A commuting group signature scheme CGS scheme satisfies traceability if for all polynomial–time adversaries $\mathcal{A}$ the advantage $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CGS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(c, c\mu, c\sigma, bpk, bsk) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU,SNDI,SIGN}}(gpk, osk)$, **if** $(bpk, bsk) \notin \mathcal{BK}$ **return** 0

$upk \leftarrow \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma))$

return 1 **if** all of the following conditions are satisfied:

$\mathsf{CGS.BlindVerify}(gpk, bpk, c, c\mu, c\sigma) = 1$ and

$\nexists uid \in \mathsf{CUL} \cup \mathsf{HUL}$ such that $\mathbf{upk}[uid] = upk$

Non–frameability. This requirement ensures that an adversary that has corrupted *all* central entities, i.e., the opener, data processors and issuer, cannot frame an honest user.

Our game requires the adversary to output blinded values but again, due to the commutative behaviour, this requirement also captures non-frameability for standard group signatures.

Definition 5.9 (CGS Non–frameability)**.** A commuting group signature scheme CGS satisfies non–frameability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{nonframe}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathbf{Exp}^{nonframe}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CGS.Setup}(1^\tau), (ipk, isk) \leftarrow\!\!\$\ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow\!\!\$\ \mathsf{CGS.OKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(uid, c^*, c\mu^*, c\sigma^*, bpk, bsk) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(gpk, isk, osk)$

return 1 **if** all of the following conditions are satisfied:

$\mathsf{CGS.BlindVerify}(gpk, bpk, c^*, c\mu^*, c\sigma^*) = 1$ and $(bpk, bsk) \in \mathcal{BK}$

$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c^*, c\mu^*, c\sigma^*)) = \mathbf{upk}[uid]$ where $uid \in \mathsf{HUL}$ and $(uid, \mathsf{CGS.UnblindM}(bsk, c^*), bpk) \notin \mathsf{SL}$

Indistinguishability of Re–randomisable Signatures. If the standard and blinded group signatures should have the additional capability to be re-randomisable, we formalise the security of that operation through an indistinguishability game, requiring that re-randomised signatures and fresh signatures are indistinguishable from each other. Such indistinguishability is used in our CLS+ security proof, given in Chapter 6, to allow for the simulation of the convert oracle.

This requirement is not implied by the anonymity and blindness requirements. Consider a construction that satisfies both blindness and anonymity. If this construction was adjusted so that standard signatures include a counter which indicates how many times it has been re-randomised (this counter would otherwise be ignored in verification, blinding and opening), the resulting construction would still satisfy both anonymity and blindness. For both the anonymity and blindness games, given an adversary $\mathcal{A}'$ that breaks the new construction, an adversary $\mathcal{A}$ could be constructed to break the original construction.

In the case of anonymity, $\mathcal{A}$ provides the same inputs to $\mathcal{A}'$ that they receive, except that a counter of 0 will be added to signatures returned by SIGN and in the guessing phase. If $\mathcal{A}'$ guesses correctly then $\mathcal{A}$ will guess correctly. In the case of blindness, $\mathcal{A}$ will provide $\mathcal{A}'$ with the same inputs in the choosing phase. The two signatures returned by $\mathcal{A}'$ will include

a counter, but this can simply be removed by $\mathcal{A}$. The resulting signature returned to $\mathcal{A}$ will be distributed identically to those in the blindness game for the new construction, as the counter is ignored by blinding. Therefore all inputs to $\mathcal{A}'$ are distributed correctly and so if $\mathcal{A}'$ guesses correctly $\mathcal{A}$ will guess correctly. Therefore a construction that clearly would not satisfy indistinguishability of re-randomised and fresh standard signatures, does satisfy anonymity and blindness. The same argument also holds for the indistinguishability of re-randomised and fresh blinded signatures.

Experiment: $\mathbf{Exp}^{rrand-b}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CGS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, opk), \mathsf{HUL}, \mathsf{CUL} \leftarrow \emptyset$

$(\mathsf{st}, uid, m, \mu, \sigma, bpk) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{USK}}(gpk, isk, osk)$

if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ or $\mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma) \neq \mathbf{upk}[uid]$ **return** 0

if $b = 0$ $(\mu', \sigma') \leftarrow_\$ \mathsf{CGS.RRand}(gpk, bpk, m, \mu, \sigma)$

if $b = 1$ $(\mu', \sigma') \leftarrow_\$ \mathsf{CGS.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{st}, \mu', \sigma')$ **return** b^*

Experiment: $\mathbf{Exp}^{blindrrand-b}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CGS.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, opk), \mathsf{HUL}, \mathsf{CUL} \leftarrow \emptyset$

$(\mathsf{st}, uid, c, c\mu, c\sigma, bpk, bsk) \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{USK}}(gpk, isk, osk)$

if $(bpk, bsk) \notin \mathcal{BK}$ or $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** 0

if $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma) \neq \mathbf{upk}[uid]$ **return** 0

if $b = 0$ $(c', c\mu', c\sigma') \leftarrow_\$ \mathsf{CGS.RRandBlind}(gpk, bpk, c, c\mu, c\sigma)$

if $b = 1$ $(c', c\mu', c\sigma') \leftarrow_\$ \mathsf{CGS.BlindSign}(gpk, bpk, \mathbf{gsk}[uid], \mathsf{CGS.UnblindM}(bsk, c))$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{st}, c', c\mu', c\sigma')$ **return** b^*

Definition 5.10 (Re–randomisable)**.** A commuting group signature scheme CGS scheme is re-randomisable if for all polynomial–time adversaries $\mathcal{A}$ the following advantages are negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{rrand-0}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1] - \Pr[\mathbf{Exp}^{rrand-1}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]\right|$$

and

$$\left|\Pr[\mathbf{Exp}^{blindrrand-0}_{\mathcal{A},\mathsf{CGS}}(\tau)=1]-\Pr[\mathbf{Exp}^{blindrrand-1}_{\mathcal{A},\mathsf{CGS}}(\tau)=1]\right|.$$

5.4 Our CGS Construction

To build our CGS–cmNIZK construction for commuting group signatures, we make use of automorphic signatures, ElGamal encryption and controlled malleable NIZKs. *Automorphic signatures* are structure-preserving signatures, for which the verification key space is contained within the message space. *Controlled-malleable NIZKs* allow proofs to be mauled to take into account a transformation of the instance, as long as this transformation is within an allowable set. This allows signatures to be blinded, but because the malleability is controlled the unforgeability properties are still satisfied.

High–level Idea. The issuer's key pair is the signing and verification key of an automorphic signature [65], which we recall in Section 5.2, and the opening key is an ElGamal encryption key pair. When joining the group, a user first generates a signing and verification key of an automorphic signature, which will be their secret and public key. The issuer then signs the user's public key to form a credential, which is possible due to the automorphic property.

When a user signs m, they encrypt their public key under the opening public key – which forms a pseudonym – and "normally" sign the message using the automorphic signature. The latter is never revealed, but gets used only in a proof of knowledge. More precisely, using a cm-NIZK, the user proves knowledge of a signature of m under its public key, an issuer's credential on its public key and correctness of the pseudonym.

During blinding, the pseudonym is re-randomised, and an extra layer of encryption under the blinding public key is added. The message is also encrypted under the blinding public key. The malleability of the cm-NIZK is then used to update the signature. The pseudonym can be opened by decrypting under the opening secret key – both in plain and blinded form. Finally, messages and opened pseudonyms can be unblinded by decrypting with the blinding secret key. The *bpk* must be fixed in signing, as it must be part of the statement proved by the cm-NIZK to allow for the proof to be transformed in blinding. This is because

cm-NIZKs are defined for relations that are closed under all allowable transformations.

Additional Structural Assumptions of Automorphic Signatures. We make the following assumptions satisfied by our concrete instantiation which we present later. The automorphic signature scheme can be simplified so either messages are elements of $\mathbb{G}_1$ and the verification key is an element of $\mathbb{G}_2$, in which case we will refer to $(\mathsf{ASetup}_1, \mathsf{AKeyGen}_1, \mathsf{ASign}_1, \mathsf{AVerify}_1)$, or messages are elements of $\mathbb{G}_2$ and the verification key is an element of $\mathbb{G}_1$, in which case we will refer to $(\mathsf{ASetup}_2, \mathsf{AKeyGen}_2, \mathsf{ASign}_2, \mathsf{AVerify}_2)$. We also assume our automorphic signatures are in the type–3 setting, ASetup takes as input the bilinear group, and that the signing key sk and verification key vk are of the form $sk \in \mathbb{Z}_p^*$ and $vk = g_j^{sk}$ when $vk \in \mathbb{G}_j$.

5.4.1 Detailed Description of our CGS–cmNIZK Construction

Setup and Key Generation. In $\mathsf{CGS.Setup}$, parameters for the automorphic signature scheme, ElGamal encryption scheme and cm-NIZKs are formed. The issuing secret and public key is set to be the signing and verification key of an automorphic signature. The opening secret and public keys are ElGamal decryption and encryption keys in $\mathbb{G}_2$, and the blinding secret and public keys are ElGamal decryption and encryption keys in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

$\mathsf{CGS.Setup}(1^\tau)$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$\, \mathcal{G}(1^\tau), \mathsf{param}_{\mathsf{auto1}} \leftarrow\!\!\$\, \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$

$\mathsf{param}_{\mathsf{auto2}} \leftarrow\!\!\$\, \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), g \leftarrow\!\!\$\, \mathbb{G}_1, \hat{g} \leftarrow\!\!\$\, \mathbb{G}_2, \sigma_{\mathsf{crs}} \leftarrow\!\!\$\, \mathsf{CRSSetup}(1^\tau)$

return $((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto1}}, \mathsf{param}_{\mathsf{auto2}}, g, \hat{g}, \sigma_{\mathsf{crs}})$

$\mathsf{CGS.IKGen}(\mathsf{param})$

$(ipk, isk) \leftarrow\!\!\$\, \mathsf{AKeyGen}_2(\mathsf{param}_{\mathsf{auto2}})$ **return** (ipk, isk)

CGS.OKGen(param)	CGS.BKGen(param)
$osk \leftarrow_\$ \mathbb{Z}_p^*, opk \leftarrow \hat{g}^{osk}$	$bsk_1, bsk_2 \leftarrow_\$ \mathbb{Z}_p^*, bpk_1 \leftarrow g^{bsk_1}, bpk_2 \leftarrow \hat{g}^{bsk_2}$
return (opk, osk)	**return** $((bpk_1, bpk_2), (bsk_1, bsk_2))$

Join. To join the group, the user first generates a key pair (usk, upk) for an automorphic signature and then obtains an issuer's signature on their public key upk. The user must first prove knowledge of usk with an SPK. However, the knowledge soundness of of this SPK is not required in any of our security proofs. This is because our security model, like the standard model for dynamic group signatures [14], ensures that even the issuer cannot frame an honest user. Therefore if an adversary could complete a join protocol fully on behalf of another user, without knowing the associated secret key, they still cannot forge signatures. However, even if impersonating another user in the join protocol does not lead to a forgery, it could still lead to a denial of service attack, and so we include the SPK.

CGS.UKGen(param)

return $(upk, usk) \leftarrow_\$ \mathsf{AKeyGen}_1(\mathsf{param}_{\mathsf{auto}1})$

$\mathcal{U}$.CGS.Join(gpk, usk, upk)	$\rightleftharpoons$	$\mathcal{I}$.CGS.Issue(isk, gpk, upk)
	$\xleftarrow{n}$	choose $n \leftarrow_\$ \{0,1\}^\tau$
$\pi_{\mathsf{join}} \leftarrow_\$ \mathsf{SPK}\{usk : upk = g_2^{usk}\}(n)$	$\xrightarrow{\pi_{\mathsf{join}}}$	
		Verify π_{join} with respect to upk
		$cred \leftarrow_\$ \mathsf{ASign}_2(isk, upk)$
	$\xleftarrow{cred}$	
check that $\mathsf{AVerify}_2(cred, ipk, upk) = 1$		
$upk' \leftarrow g_1^{usk}$ **return** $\mathbf{gsk}[uid] \leftarrow (usk, upk, upk', cred)$,		

Figure 5.4: Join protocol of our CGS–cmNIZK construction

Sign and Verification of standard group signatures. When signing a message m under the signing key $\mathbf{gsk}[uid] = (usk, upk, upk', cred)$, the user's public key is encrypted under the opening public key. By setting $\mu_2 = 1$, μ can also be seen as an encryption

under the blinding key with encryption randomness 0. The tuple $(1, m)$ can also be seen as an encryption of m under the blinding key with encryption randomness 0. This is necessary so that blinding encryption can be added by "re-randomising" these ciphertexts in CGS.Blind, while maintaining the capability to update the associated proof. This is because cm-NIZKs are defined for relations that are closed under all allowable transformations and so encryption under the blinding key must be included during signing to allow for the later blinding of the signature. The user then signs m with their user secret key to output Ω.

The signature is never output, but instead a cm-NIZK is computed which proves that μ is an encryption of upk, c is an "encryption" of m (with randomness 0), and knowledge of a correct Ω. The latter comprises showing that Ω is a valid signature on m under upk, and knowledge of a membership credential $cred$ that is a valid signature on upk under ipk.

More formally we define the relation R such that $((opk, bpk_1, bpk_2, ipk, \mu, c)$, $(upk', upk, cred, \Omega, g_1^{\alpha}, g_1^{\beta}, g_2^{\gamma}, m)) \in R$ if and only if:

$$e(g_1, \mu_1) = e(g_1^{\alpha}, \hat{g}), \quad e(g_1, \mu_2) = e(g_1^{\beta}, \hat{g}), \quad \text{and} \tag{5.1}$$

$$e(g_1, \mu_3) = e(g_1, upk)e(g_1^{\alpha}, opk)e(g_1^{\beta}, bpk_2), \quad \text{and} \tag{5.2}$$

$$\mathsf{AVerify}_1(\Omega, upk, m) = 1, \quad \mathsf{AVerify}_2(cred, ipk, upk) = 1, \quad \text{and} \tag{5.3}$$

$$e(c_1, g_2) = e(g, g_2^{\gamma}), \quad e(c_2, g_2) = e(m, g_2)e(bpk_1, g_2^{\gamma}), \text{ and} \tag{5.4}$$

$$e(upk', g_2) = e(g_1, upk). \tag{5.5}$$

These checks ensure that a pseudonym is the ElGamal encryption of the upk under the blinding and opening key (5.1,5.2), the ciphertext c is an ElGamal encryption of the message under the blinding key (5.4), Ω is a valid signature under the user public key on the message (5.3), the credential is a valid signature under the issuing public key on the user public key (5.3), and that $upk' = g_1^{usk}$ (5.5).

We define the allowable set of transformations to be:

$$\mathcal{T} = \{(r_{\mathsf{enc1}}, r_{\mathsf{enc2}}, r_{\mathsf{enc3}}) : r_{\mathsf{enc1}}, r_{\mathsf{enc2}}, r_{\mathsf{enc3}} \in \mathbb{Z}_p\},$$

such that for $T = (r_{\mathsf{enc1}}, r_{\mathsf{enc2}}, r_{\mathsf{enc3}})$, the transformation

$T_{\mathsf{inst}}(opk, bpk, ipk, \mu, c) =$

$$(opk, bpk, ipk, (\mu_1 \hat{g}^{r_{\mathsf{enc1}}}, \mu_2 \hat{g}^{r_{\mathsf{enc2}}}, \mu_3 \cdot opk^{r_{\mathsf{enc1}}} bpk_2^{r_{\mathsf{enc2}}}), (c_1 g^{r_{\mathsf{enc3}}}, c_2 bpk_1^{r_{\mathsf{enc3}}})),$$

and $T_{\mathsf{wit}}(upk', upk, cred, \Omega, g_1^{\alpha}, g_1^{\beta}, g_2^{\gamma}, m) =$

$$(upk', upk, cred, \Omega, g_1^{\alpha} g_1^{r_{\mathsf{enc1}}}, g_1^{\beta} g_1^{r_{\mathsf{enc2}}}, g_2^{\gamma} g_2^{r_{\mathsf{enc3}}}, m).$$

We show later that this relation and transformation can be instantiated as a cm-NIZK.

We note that upk' is only included as a witness, to allow for the *extractability* requirement necessary for the building of a CLS+ scheme, as shown in Chapter 6. This requirement ensures that if a valid blinded signature does not open to a user public key that was input to a simulator, upk' can be extracted. If this is not necessary, upk' can be removed, which will not affect the security of the scheme. In Chapter 6, we define the extractability property and show that this construction satisfies it.

In more detail, CGS.Sign and CGS.Verify are defined as follows:

CGS.Sign$(gpk, bpk, \mathbf{gsk}[uid], m)$

parse $\mathbf{gsk}[uid] = (usk, upk, upk', cred), \alpha \leftarrow_{\$} \mathbb{Z}_p^*, (\mu_1, \mu_2, \mu_3) \leftarrow (\hat{g}^{\alpha}, 1, upk \cdot opk^{\alpha})$
$\beta \leftarrow 0, \gamma \leftarrow 0, c \leftarrow (1, m), \Omega \leftarrow_{\$} \mathsf{ASign}_1(usk, m)$
$\sigma \leftarrow_{\$}$ cm-NIZK$\{(upk', upk, cred, \Omega, g_1^{\alpha}, g_1^{\beta}, g_2^{\gamma}, m) : e(g_1, \mu_1) = e(g_1^{\alpha}, \hat{g}) \wedge$
$e(g_1, \mu_2) = e(g_1^{\beta}, \hat{g}) \wedge e(g_1, \mu_3) = e(g_1, upk) e(g_1^{\alpha}, opk) e(g_1^{\beta}, bpk_2) \wedge$
$\mathsf{AVerify}_1(\Omega, upk, m) = 1 \wedge \mathsf{AVerify}_2(cred, ipk, upk) = 1 \wedge e(c_1, g_2) = e(g, g_2^{\gamma}) \wedge$
$e(c_2, g_2) = e(m, g_2) e(bpk_1, g_2^{\gamma}) \wedge e(upk', g_2) = e(g_1, upk)\}$ **return** $((\mu_1, \mu_2, \mu_3), \sigma)$

CGS.Verify$(gpk, bpk, m, \mu, \sigma)$

Check $\mu_2 = 1$, Verify σ with respect to $(opk, bpk, ipk, \mu, (1, m))$

Blinding and blind verification. During blinding, the pseudonym and message are encrypted under the blinding public key, and the encryption under the opening public

key is re-randomised, so that this encryption randomness cannot be used to unblind pseudonyms. The cm-NIZK is transformed with ZKEval so that it is consistent with the blinded pseudonym, and message. When ZKEval is performed, the proof is also re-randomised due to the derivation privacy property. In CGS.BlindSign, the pseudonym is encrypted under the blinding and opening public key, and the message is encrypted under the blinding public key. The proof cm-NIZK is then computed as in CGS.Sign.

CGS.BlindSign$(gpk, bpk, \mathbf{gsk}[uid], m)$

parse $\mathbf{gsk}[uid] = (usk, upk, upk', cred), \alpha, \beta, \gamma \leftarrow\!\!\$\ \mathbb{Z}_p^*$

$c\mu \leftarrow (\hat{g}^{\alpha}, \hat{g}^{\beta}, upk \cdot opk^{\alpha} bpk_2^{\beta}), c \leftarrow (g^{\gamma}, m \cdot bpk_1^{\gamma}), \Omega \leftarrow\!\!\$\ \mathsf{ASign}_1(usk, m)$

$c\sigma \leftarrow\!\!\$\ \mathsf{cm\text{-}NIZK}\{(upk', upk, cred, \Omega, g_1^{\alpha}, g_1^{\beta}, g_2^{\gamma}, m) : e(g_1, c\mu_1) = e(g_1^{\alpha}, \hat{g}) \wedge$

$e(g_1, c\mu_2) = e(g_1^{\beta}, \hat{g}) \wedge e(g_1, c\mu_3) = e(g_1, upk) e(g_1^{\alpha}, opk) e(g_1^{\beta}, bpk_2) \wedge$

$\mathsf{AVerify}_1(\Omega, upk, m) = 1 \wedge \mathsf{AVerify}_2(cred, ipk, upk) = 1 \wedge e(c_1, g_2) = e(g, g_2^{\gamma}) \wedge$

$e(c_2, g_2) = e(m, g_2) e(bpk_1, g_2^{\gamma}) \wedge e(upk', g_2) = e(g_1, upk)\}$

return $(c, c\mu, c\sigma)$

CGS.Blind$(gpk, bpk, m, \mu, \sigma)$

if $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) = 0$ **return** $\perp$

$\alpha', \beta', \gamma' \leftarrow\!\!\$\ \mathbb{Z}_p^*, c\mu \leftarrow (\mu_1 \hat{g}^{\alpha'}, \mu_2 \hat{g}^{\beta'}, \mu_3 opk^{\alpha'} bpk_2^{\beta'}), c \leftarrow (g^{\gamma'}, m \cdot bpk_1^{\gamma'})$

$c\sigma \leftarrow\!\!\$\ \mathsf{ZKEval}(\sigma_{\mathsf{crs}}, (\alpha', \beta', \gamma'), (opk, bpk, ipk, \mu, (1, m)), \sigma)$, **return** $(c, c\mu, c\sigma)$

CGS.BlindVerify$(gpk, bpk, c, c\mu, c\sigma)$

Check $c\sigma$ with respect to $c\mu, c, gpk$ and bpk

CGS.BlindUser(bpk, upk)

$\beta \leftarrow\!\!\$\ \mathbb{Z}_p^*$ **return** $cupk \leftarrow (\hat{g}^{\beta}, upk \cdot bpk_2^{\beta})$

Opening and unblinding. The opener first checks the validity of the provided input and decrypts the standard or blinded pseudonym under its opening secret key *osk*. It outputs the decrypted value, which is either the plain user public key or an encryption thereof.

$\underline{\mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma)}$

if $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) = 0$

  return $\perp$

return $upk \leftarrow \mu_3 \mu_1^{-osk}$

$\underline{\mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma)}$

if $\mathsf{CGS.BlindVerify}(gpk, bpk, c, c\mu, c\sigma) = 0$

  return $\perp$

return $cupk \leftarrow (c\mu_2, c\mu_3 \cdot c\mu_1^{-osk})$

To unblind messages and user public keys, they are simply decrypted via bsk.

$\underline{\mathsf{CGS.UnblindUser}(bsk, cupk)}$

return $upk \leftarrow cupk_2 cupk_1^{-bsk_2}$

$\underline{\mathsf{CGS.UnblindM}(bsk, c)}$

return $c_2 c_1^{-bsk_1}$

Re-randomisation. In order to re-randomise both standard and blinded signatures, firstly the encryption under both the opening and blinding public key is re-randomised. The cm-NIZK is then updated with ZKEval to take this into account.

$\underline{\mathsf{CGS.RRand}(gpk, bpk, m, \mu, \sigma)}$

if $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) = 0$ **return** $\perp, \alpha' \leftarrow\!\!\$\, \mathbb{Z}_p^*, \mu' \leftarrow (\mu_1 \hat{g}^{\alpha'}, \mu_2, \mu_3 opk^{\alpha'})$

$\sigma' \leftarrow\!\!\$\, \mathsf{ZKEval}(\sigma_{\mathsf{crs}}, (\alpha', 0, 0), (opk, bpk, ipk, \mu, (1, m)), \sigma)$, **return** (μ', σ')

$\underline{\mathsf{CGS.RRandBlind}(gpk, bpk, c, c\mu, c\sigma)}$

if $\mathsf{CGS.BlindVerify}(gpk, bpk, c, c\mu, c\sigma) = 0$ **return** $\perp, \alpha', \beta', \gamma' \leftarrow\!\!\$\, \mathbb{Z}_p^*$

$c\mu' \leftarrow (c\mu_1 \hat{g}^{\alpha'}, c\mu_2 \hat{g}^{\beta'}, c\mu_3 opk^{\alpha'} bpk_2^{\beta'}), c' \leftarrow (c_1 g^{\gamma'}, c_2 bpk_1^{\gamma'})$

$c\sigma' \leftarrow\!\!\$\, \mathsf{ZKEval}(\sigma_{\mathsf{crs}}, (\alpha', \beta', \gamma'), (opk, bpk, ipk, c\mu, c), c\sigma)$, **return** $(c', c\mu', c\sigma')$

5.5 Security of our CGS–cmNIZK construction

We now show that our CGS–cmNIZK construction satisfies the security properties for commuting group signatures defined in Section 5.3.2; i.e., the following theorem holds.

Theorem 5.1. The CGS–cmNIZK construction presented in Section 5.4 is a secure CGS as defined in Section 5.3.2 if

- the automorphic signatures schemes are EUF-cma secure and satisfy the additional structural assumptions given in Section 5.4,
- the cm-NIZK is zero–knowledge, strongly derivation private and controlled-malleable simulation-sound extractable (cm-SSE),
- the SPK (used in Join) is a zero-knowledge proof of knowledge,
- the DDH assumptions holds in $\mathbb{G}_1$ and $\mathbb{G}_2$.

We now show that our CGS–cmNIZK construction satisfies the correctness, commutative behaviour, re-randomisability, traceability, non-frameability, anonymity and blindness requirements given in Section 5.3.2.

5.5.1 Correctness

The first twos conditions are clearly satisfied due to the correctness of the cm-NIZKs used, and because if $\mu = (\hat{g}^{\alpha}, 1, upk \cdot opk^{\alpha})$, then $\mu_3 \mu_1^{-osk} = upk$. Due to the correctness of the cm-NIZKs used, the transformed proof will still be valid and so the signature will still be valid after blinding. As $c = (g^{\gamma'}, m \cdot bpk_1^{\gamma'})$, we have $\mathsf{CGS.UnblindM}(bsk, c) = c_2 c_1^{-bsk_1} = m$. Because $upk = \mu_3 \mu_1^{-osk}$ and $\mu_2 = 1$, we have $c\mu = (\mu_1 \hat{g}^{\alpha}, \hat{g}^{\beta}, \mu_3 opk^{\alpha} bpk_2^{\beta})$. Therefore,

$$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma))$$

$$= \mathsf{CGS.UnblindUser}(bsk, (\hat{g}^{\beta}, \mu_3 bpk_2^{\beta} \mu_1^{-osk})) = \mu_3 \mu_1^{-osk} = upk.$$

5.5.2 Commutative Behaviour

CGS.BlindSign is indistinguishable from CGS.Sign followed by CGS.Blind because $c, c\mu$ are distributed identically when they are computed in CGS.Blind with input a valid signature, to when they are computed in CGS.BlindSign. A polynomial–time adversary cannot distinguish between $c\sigma$ generated in CGS.Blind or CGS.BlindSign due to the derivation privacy of cm-NIZKs.

CGS.BlindUser is indistinguishable from the output of CGS.OpenBlind on a valid blinded

signature with the same user public key, because the adversary must return μ of the form $(\hat{g}^{\alpha}, 1, upk \cdot opk^{\alpha})$, which will be of the form $(\hat{g}^{\alpha+\alpha'}, \hat{g}^{\beta}, upk \cdot opk^{\alpha+\alpha'} bpk_2^{\beta})$ after blinding, and $(\hat{g}^{\beta}, upk \cdot bpk_2^{\beta})$ after opening. This is identically distributed to CGS.BlindUser on input upk.

For the final requirement, if the adversary outputs an invalid signature, then CGS.Blind will also fail. If the adversary outputs a valid signature, then due to the correctness of the cm-NIZK used, CGS.Blind will output a valid proof, and so CGS.BlindVerify will output 1. Because $upk = \mu_3 \mu_1^{-osk}$ and $\mu_2 = 1$, $c\mu = (\mu_1 \hat{g}^{\alpha}, \hat{g}^{\beta}, \mu_3 opk^{\alpha} bpk_2^{\beta})$. Then

$$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma))$$

$$= \mathsf{CGS.UnblindUser}(bsk, (\hat{g}^{\beta}, \mu_3 bpk_2^{\beta} \mu_1^{-osk})) = \mu_3 \mu_1^{-osk} = upk.$$

5.5.3 Re-randomisability

The pseudonyms output by CGS.Sign and CGS.RRand and the pseudonyms and blinded messages output by CGS.BlindSign and CGS.RRandBlind are identically distributed. The adversary cannot distinguish between a fresh cm-NIZK and a transformed one due to the derivation privacy property. Therefore, the CGS–cmNIZK construction is re-randomisable.

5.5.4 Anonymity

Lemma 5.1. The CGS–cmNIZK construction presented in Section 5.4 satisfies **anonymity** if the DDH assumption holds in $\mathbb{G}_2$, and the cm-NIZK is zero–knowledge and cm-SSE.

Proof. First we show that if an adversary $\mathcal{A}$ exists, such that

$$|\Pr[\mathbf{Exp}^{anon-0}_{\mathcal{A},\mathsf{CGS–cmNIZK}}(\tau) = 1] - \Pr[\mathbf{Exp}^{anon-join-1}_{\mathcal{A},\mathsf{CGS–cmNIZK}}(\tau) = 1]| = \epsilon,$$

where ϵ is non-negligible, then we can can build an adversary $\mathcal{A}'$ that distinguishes DDH tuples with non-negligible probability. We provide $\mathcal{A}'$ in Figure 5.5. We then describe why the simulation given in Figure 5.5 and the anonymity experiment are indistinguishable

towards $\mathcal{A}$, provided $\mathcal{A}'$ is input a DDH tuple, and otherwise that $\mathcal{A}$ guesses correctly with probability 1/2.

The simulator SE_1 outputs a σ_{crs} that is identical to CRSSetup. The values $\hat{g}, opk$ are distributed identically to the anonymity experiment. Therefore, gpk, isk are distributed identically to the anonymity experiment. Assuming a DDH tuple is input, the inputs to $\mathcal{A}$ in the guessing phase are identically distributed. This is because $\mu^* = (D_3, 1, D_4\mathbf{upk}[uid_b^*])$ is identically distributed to CGS.Sign, as $\log_{D_3} D_4 = \log_{\hat{g}} opk$. The ciphertext c^* is generated identically to CGS.Sign. The proof σ^* can be simulated due to the zero–knowledge property of the cm-NIZK used.

Simulating the Oracles. The SNDU and SIGN oracles are identical to the anonymity experiment.

The OPEN and BOPEN oracles are identically distributed, provided they successfully extract upk and do not abort.

The only case in which they will not successfully extract upk is the case where instead the extraction algorithm outputs a statement x', and a transformation T, where $x' = (opk, bpk^*, ipk, \mu^*, c^*)$, as this is the only statement for which a simulation is generated, and $x = T_x(x')$ and T is a valid transform. This means that $c\mu/\ \mu$, and $c/\ (1, m)$, are re-randomisations of μ^*, c^*, and so $c/\ (1, m)$ is an encryption of m^*, and $c\mu/\ \mu$ are encryptions of $\mathbf{upk}[uid_b^*]$. Also $bpk = bpk^*$. In this case the OPEN and BOPEN oracles would have aborted anyway.

Both oracles will also abort due to invalid inputs identically to the anonymity game. In BOPEN, the blinding encryption randomness is preserved by outputting $(c\mu_2, upkc\mu_2^{bsk_2})$ which is identically distributed to the output of CGS.OpenBlind.

Reduction to the DDH problem. If a DDH tuple is not input, D_4 was chosen randomly, and so the input to $\mathcal{A}$ is independent of b. Therefore $\mathcal{A}$ guesses correctly with probability 1/2.

SNDU(uid, n)

Identical to the anonymity experiment.

SIGN(uid, m, bpk)

Identical to the anonymity experiment.

OPEN(m, μ, σ, bpk)

if $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma) = 0$ **return** $\perp$
$((\cdot, upk, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot), \cdot, \cdot) \leftarrow E_2(\sigma_{\mathsf{crs}}, \tau_e, (opk, bpk, ipk, \mu, (1, m)), \sigma)$
if $upk = \perp$ **return** $\perp$
if $upk = \mathbf{upk}[uid_d^*]$ s.t $d \in \{0, 1\}$ and $m = m^*$ and $bpk = bpk^*$ **return** $\perp$
else return upk

BOPEN$(c, c\mu, c\sigma, bpk, bsk)$

if $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
if $\mathsf{CGS.BlindVerify}(gpk, bpk, c, c\mu, c) = 0$ **return** $\perp$
$((\cdot, upk, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot), \cdot, \cdot) \leftarrow E_2(\sigma_{\mathsf{crs}}, \tau_e, (opk, bpk, ipk, c\mu, c), c\sigma)$
$m \leftarrow \mathsf{CGS.UnblindM}(bsk, c)$
if $upk = \perp$ **return** $\perp$
if $upk = \mathbf{upk}[uid_d]$ s.t $d \in \{0, 1\}$ and $m = m^*$ and $bpk = bpk^*$ **return** $\perp$
else return $(c\mu_2, upkc\mu_2^{bsk_2})$

$\mathcal{A}'(D_1, D_2, D_3, D_4)$

$b \leftarrow\!\!\$\ \{0, 1\}, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$\ \mathcal{G}(1^\tau)$
$\mathsf{param}_{\mathsf{auto}1} \leftarrow\!\!\$\ \mathsf{ASetup}_1(1^\tau), \mathsf{param}_{\mathsf{auto}2} \leftarrow\!\!\$\ \mathsf{ASetup}_2(1^\tau)$
$g \leftarrow\!\!\$\ \mathbb{G}_1, \hat{g} \leftarrow D_1, (\sigma_{\mathsf{crs}}, \tau_s, \tau_e) \leftarrow\!\!\$\ \mathsf{SE}_1(1^\tau),$
$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$
$(ipk, isk) \leftarrow\!\!\$\ \mathsf{CGS.IKGen}(\mathsf{param}), opk \leftarrow D_2, gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(\mathsf{st}, uid_0^*, uid_1^*, m^*, bpk^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU,SIGN,OPEN,BOPEN}}(\mathsf{choose}, gpk, isk)$
if $uid_0^*, uid_1^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0^*], \mathbf{gsk}[uid_1^*] = \perp$ $b^* \leftarrow 0$
  if $b^* = b$ **return** 1 **else return** 0
$\mu^* \leftarrow (D_3, 1, D_4\mathbf{upk}[uid_b^*]), c^* \leftarrow (1, m^*)$
$\sigma^* \leftarrow\!\!\$\ S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk^*, ipk, \mu^*, c^*))$
$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU,SIGN,OPEN,BOPEN}}(\mathsf{guess}, \mathsf{st}, \mu^*, \sigma^*)$
if $b^* = b$ **return** 1 **else return** 0

Figure 5.5: $\mathcal{A}'$ which distinguishes DDH tuples in $\mathbb{G}_2$ using $\mathcal{A}$ which breaks the anonymity of CGS–cmNIZK with probability ϵ

The probability $\mathcal{A}$ is successful is:

$$1/2\Pr[\mathbf{Exp}^{anon-0}_{\mathcal{A},\Pi}(\tau)=0]+1/2\Pr[\mathbf{Exp}^{anon-1}_{\mathcal{A},\Pi}(\tau)=1]=(\epsilon+1)/2.$$

Therefore $\mathcal{A}'$ has an $\epsilon/2$ advantage in distinguishing DDH tuples.

□

5.5.5 Blindness

Lemma 5.2. The CGS–cmNIZK construction presented in Section 5.4 satisfies **blindness** if the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$, and the cm-NIZK is zero–knowledge and strongly derivation private.

Proof. In this proof we will show that the following Games 0 -5 are indistinguishable. As Game 0 is the blindness game with the CGS–cmNIZK construction, and in Game 5 all inputs to the adversary are independent of the bit they must guess, our CGS–cmNIZK construction satisfies blindness. We give a summary of the games in Table 5.1.

Game	Change	Indistinguishability to previous game
Game 0	The blindness game with the CGS–cmNIZK construction.	
Game 1	σ_{crs} generated by S_1 instead of CRSSetup.	Outputs of S_1 and CRSSetup identically distributed.
Game 2	$c\sigma^*$ generated by S_2 instead of ZKEval.	Due to the strong derivation privacy of the cm-NIZK.
Game 3	$c\mu^*$ chosen randomly.	Due to the DDH assumption which implies the security of Elgamal encryption.
Game 4	c^* chosen randomly.	Due to the DDH assumption which implies the security of Elgamal encryption.
Game 5	*cupk* chosen randomly.	Due to the DDH assumption which implies the security of Elgamal encryption.

Table 5.1: Games in our blindness proof

We define Game 0 to be the blindness experiment with the commuting group signature construction. Let P_0 be the event that an adversary $\mathcal{A}$ correctly guesses b after Game 0.

We define Game 1 to be identical to Game 0, except that σ_{crs} is generated by S_1 instead of CRSSetup, and a simulation trapdoor is also generated. As outputs of S_1 and CRSSetup are identically distributed, letting P_1 be the event that the adversary $\mathcal{A}$ correctly guesses b after Game 1, then $\Pr[P_0] = \Pr[P_1]$.

We define Game 2 to be identical to Game 1, except during blinding instead of transforming the proof with ZKEval instead the proof is simulated. Let P_2 be the event that the adversary $\mathcal{A}$ correctly guesses b after Game 2.

Game 2

$b \leftarrow\$ \{0,1\}, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\$ \mathcal{G}(1^\tau), g \leftarrow\$ \mathbb{G}_1, \hat{g} \leftarrow\$ \mathbb{G}_2$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow\$ \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$

$\mathsf{param}_{\mathsf{auto}2} \leftarrow\$ \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow\$ S_1(1^\tau)$

$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$

$(ipk, isk) \leftarrow\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$(bpk, bsk) \leftarrow\$ \mathsf{CGS.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(\mathsf{st}, (upk_0, \mu_0, \sigma_0, m_0), (upk_1, \mu_1, \sigma_1, m_1)) \leftarrow\$ \mathcal{A}(\mathsf{choose}, gpk, bpk, isk, osk)$

if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0$ **return** 0

$cupk \leftarrow\$ \mathsf{CGS.BlindUser}(bpk, upk_b)$

$\alpha, \beta, \gamma \leftarrow\$ \{0,1\}^*, c\mu^* \leftarrow (\mu_{b,1}\hat{g}^\alpha, \mu_{b,2}\hat{g}^\beta, \mu_{b,3} opk^\alpha bpk_2^\beta),$

$c^* \leftarrow (g^\gamma, m_b bpk_1^\gamma), c\sigma^* \leftarrow\$ S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, c\mu^*, c^*))$

$b^* \leftarrow\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, cupk, c^*, c\mu^*, c\sigma^*)$ **return** b^*

We show that Game 1 and Game 2 are indistinguishable, assuming the cm-NIZK proof is strongly derivation private. We provide a distinguishing algorithm $\mathcal{D}_1$ in Figure 5.6 that aims to guess b' in the Strongly Derivation Private security game.

We now show that when $b' = 1$, inputs to $\mathcal{A}$ are identical to Game 1, and when $b' = 0$ inputs to $\mathcal{A}$ are identical to Game 2.

The gpk, isk, osk input to $\mathcal{A}$ is the same as in both Game 1 and Game 2. The challenge pseudonym and ciphertext $c\mu^*$ and c^* are re-randomisations of $c\mu_b$ and c_b, which is identical to the CGS.Blind algorithm, and therefore identical to both Game 1 and Game 2. The blinded user public key $cupk$ is computed identically to both Game 1 and Game 2.

$\mathcal{D}_1(\sigma_{\mathsf{crs}})$

$b \leftarrow\!\!\$\, \{0,1\}, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$\, \mathcal{G}(1^\tau)$
$\mathsf{param}_{\mathsf{auto1}} \leftarrow\!\!\$\, \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$
$\mathsf{param}_{\mathsf{auto2}} \leftarrow\!\!\$\, \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow\!\!\$\, S_1(1^\tau)$
$g \leftarrow\!\!\$\, \mathbb{G}_1, \hat{g} \leftarrow\!\!\$\, \mathbb{G}_2, \mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto1}}, \mathsf{param}_{\mathsf{auto2}}, g, \hat{g}, \sigma_{\mathsf{crs}})$
$(ipk, isk) \leftarrow\!\!\$\, \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow\!\!\$\, \mathsf{CGS.OKGen}(\mathsf{param})$
$(bpk, bsk) \leftarrow\!\!\$\, \mathsf{CGS.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(\mathsf{st}_1, (upk_0, \mu_0, \sigma_0, m_0), (upk_1, \mu_1, \sigma_1, m_1)) \leftarrow\!\!\$\, \mathcal{A}(\mathsf{choose}, gpk, bpk, isk, osk)$
if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0 \quad b^* \leftarrow 0$
  if $b^* = b$ **return** 1 **else return** 0
$\alpha, \beta, \gamma \leftarrow\!\!\$\, \{0,1\}^*$
return $(\mathsf{st}_2, (opk, bpk, ipk, \mu_b, (1, m_b)), \sigma_b, (\alpha, \beta, \gamma))$

$\mathcal{D}_1(\mathsf{st}_2, \pi^*)$

$c\mu_1^* \leftarrow \mu_{b,1}\hat{g}^\alpha, c\mu_2^* \leftarrow \mu_{b,2}\hat{g}^\beta, c\mu_3^* \leftarrow \mu_{b,3} opk^\alpha bpk_2^\beta$
$c_1^* \leftarrow g^\gamma, c_2^* \leftarrow m_b bpk_1^\gamma, cupk \leftarrow\!\!\$\, \mathsf{CGS.BlindUser}(bpk, upk_b)$
$b^* \leftarrow\!\!\$\, \mathcal{A}(\mathsf{guess}, \mathsf{st}_1, cupk, c^*, c\mu^*, \pi^*), \quad$ **if** $b^* = b \quad$ **return** 1

Figure 5.6: $\mathcal{D}_1$ that distinguishes between Game 1 and Game 2 in the CGS blindness proof

If $b' = 0$, $\mathcal{D}_1$ is returned with the simulation of a proof for the statement

$$T(opk, bpk, ipk, \mu_b, (1, m_b)) = (opk, bpk, ipk, c\mu^*, c^*),$$

which is identical to Game 2.

If $b' = 1$, $\mathcal{D}_1$ is returned with the transformation, defined by (α, β, γ), of the proof. This is identical to Game 1.

Therefore $|\Pr[P_1] - \Pr[P_2]| \leq \epsilon_{sdp}$, where ϵ_{sdp} is the advantage in breaking the strong derivation privacy of the cm-NIZK.

We define Game 3 to be identical to Game 2, except $c\mu^*$ is chosen randomly. Let P_3 be the event that the adversary $\mathcal{A}$ correctly guesses b after Game 3.

Game 3

$b \leftarrow_\$ \{0,1\}, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow_\$ \mathcal{G}(1^\tau), g \leftarrow_\$ \mathbb{G}_1, \hat{g} \leftarrow_\$ \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow_\$ S_1(1^\tau)$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow_\$ \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow_\$ \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$

$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$

$(ipk, isk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$(bpk, bsk) \leftarrow_\$ \mathsf{CGS.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(\mathsf{st}, (upk_0, \mu_0, \sigma_0, m_0), (upk_1, \mu_1, \sigma_1, m_1)) \leftarrow_\$ \mathcal{A}(\mathsf{choose}, gpk, bpk, isk, osk)$

if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0$ **return** 0

$\gamma \leftarrow_\$ \{0,1\}^*, c\mu^* \leftarrow_\$ \mathbb{G}_2^3, c_1^* \leftarrow g^\gamma, c_2^* \leftarrow m_b bpk_1^\gamma$

$c\sigma^* \leftarrow_\$ S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, c\mu^*, c^*)), cupk \leftarrow_\$ \mathsf{CGS.BlindUser}(bpk, upk_b)$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, cupk, c^* c\mu^*, c\sigma^*)$ **return** b^*

We define Game 4 to be identical to Game 3, except c^* is chosen randomly. Let P_4 be the event that the adversary $\mathcal{A}$ correctly guesses b after Game 4.

Game 4

$b \leftarrow_\$ \{0,1\}, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow_\$ \mathcal{G}(1^\tau), g \leftarrow_\$ \mathbb{G}_1, \hat{g} \leftarrow_\$ \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow_\$ S_1(1^\tau)$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow_\$ \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow_\$ \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$

$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$

$(ipk, isk) \leftarrow_\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow_\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$(bpk, bsk) \leftarrow_\$ \mathsf{CGS.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(\mathsf{st}, (upk_0, \mu_0, \sigma_0, m_0), (upk_1, \mu_1, \sigma_1, m_1)) \leftarrow_\$ \mathcal{A}(\mathsf{choose}, gpk, bpk, isk, osk)$

if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0$ **return** 0

$c\mu^* \leftarrow_\$ \mathbb{G}_2^3, c^* \leftarrow_\$ \mathbb{G}_1^2$

$c\sigma^* \leftarrow_\$ S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, c\mu^*, c^*)), cupk \leftarrow_\$ \mathsf{CGS.BlindUser}(bpk, upk_b)$

$b^* \leftarrow_\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, cupk, c^*, c\mu^*, c\sigma^*)$ **return** b^*

We define Game 5 to be identical to Game 4, except *cupk* is chosen randomly. Let P_5 be the event that the adversary $\mathcal{A}$ correctly guesses b after Game 5. Clearly the probability that $\mathcal{A}$ correctly guesses b is $1/2$, as all inputs are now independent of b. Therefore $\Pr[P_5] = 1/2$.

$\mathcal{D}_2(D_1, D_2, D_3, D_4)$

$b \leftarrow\!\!\$ \{0,1\}, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$ \mathcal{G}(1^\tau), g \leftarrow\!\!\$ \mathbb{G}_1, \hat{g} \leftarrow D_1, (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow\!\!\$ S_1(1^\tau)$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow\!\!\$ \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow\!\!\$ \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$

$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$

$(ipk, isk) \leftarrow\!\!\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow\!\!\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$bsk_1 \leftarrow\!\!\$ \mathbb{Z}_p^*, bpk_1 \leftarrow g^{bsk_1}, bpk_2 \leftarrow D_2, gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(\mathsf{st}, (\mu_0, \sigma_0, m_0), (\mu_1, \sigma_1, m_1)) \leftarrow\!\!\$ \mathcal{A}(\mathsf{choose}, gpk, (bpk_1, bpk_2), isk, osk)$

if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0 \quad b^* \leftarrow 0$

  if $b^* = b$ **return** 1 **return** 0

$\gamma, \beta \leftarrow\!\!\$ \mathbb{Z}_p^*, c\mu^* \leftarrow (\mu_{b,1}\hat{g}^\beta, D_3, \mu_{b,3} D_4 opk^\beta), c_1^* \leftarrow g^\gamma, c_2^* \leftarrow m_b bpk_1^\gamma$

$c\sigma^* \leftarrow\!\!\$ S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, c\mu^*, c^*))$

$b^* \leftarrow\!\!\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, c^*, c\mu^*, c\sigma^*) \quad$ **if** $b^* = b \quad$ **return** $1 \quad$ **return** 0

Figure 5.7: $\mathcal{D}_2$ that distinguishes between Game 2 and Game 3 in the CGS blindness proof

Game 5

$b \leftarrow\!\!\$ \{0,1\}, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$ \mathcal{G}(1^\tau), g \leftarrow\!\!\$ \mathbb{G}_1, \hat{g} \leftarrow\!\!\$ \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow\!\!\$ S_1(1^\tau)$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow\!\!\$ \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow\!\!\$ \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$

$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$

$(ipk, isk) \leftarrow\!\!\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow\!\!\$ \mathsf{CGS.OKGen}(\mathsf{param})$

$(bpk, bsk) \leftarrow\!\!\$ \mathsf{CGS.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(\mathsf{st}, (upk_0, \mu_0, \sigma_0, m_0), (upk_1, \mu_1, \sigma_1, m_1)) \leftarrow\!\!\$ \mathcal{A}(\mathsf{choose}, gpk, bpk, isk, osk)$

if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0 \quad$ **return** 0

$c\mu^* \leftarrow\!\!\$ \mathbb{G}_2^3, c^* \leftarrow\!\!\$ \mathbb{G}_1^2, cupk \leftarrow\!\!\$ \mathbb{G}_2^2$

$c\sigma^* \leftarrow\!\!\$ S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, c\mu^*, c^*))$

$b^* \leftarrow\!\!\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, cupk, c^*, c\mu^*, c\sigma^*) \qquad$ **return** b^*

We show that Game 2 and Game 3 are indistinguishable assuming the DDH assumption in $\mathbb{G}_2$. We provide a distinguishing algorithm $\mathcal{D}_2$ in Figure 5.7.

If $\mathcal{D}_2$ is input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 2. This is because, letting $\alpha = \log_{\hat{g}} D_3$, then $D_4 = bpk_2^\alpha$, and therefore $c\mu^*$ is distributed identically to Game 2.

If $\mathcal{D}_2$ is not input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 3. This is because, β, D_3, D_4 are now chosen independently and randomly.

Therefore $|\Pr[P_2] - \Pr[P_3] \leq \epsilon_{\mathsf{DDH}}$, where ϵ_{DDH} is the DDH advantage, and therefore

$\mathcal{D}_3(D_1, D_2, D_3, D_4)$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\$ \mathcal{G}(1^\tau), g \leftarrow D_1, \hat{g} \leftarrow\$ \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow\$ S_1(1^\tau)$
$\mathsf{param}_{\mathsf{auto}1} \leftarrow\$ \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow\$ \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$
$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$
$(ipk, isk) \leftarrow\$ \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow\$ \mathsf{CGS.OKGen}(\mathsf{param})$
$bsk_2 \leftarrow\$ \mathbb{Z}_p^*, bpk_2 \leftarrow \hat{g}^{bsk_2}, bpk_1 \leftarrow D_2, gpk \leftarrow (\mathsf{param}, ipk, opk),$
$(\mathsf{st}, (\mu_0, \sigma_0, m_0), (\mu_1, \sigma_1, m_1)) \leftarrow\$ \mathcal{A}(\mathsf{choose}, gpk, (bpk_1, bpk_2), isk, osk)$
if $\exists d \in \{0,1\}$ s.t $\mathsf{CGS.Verify}(gpk, bpk, m_d, \mu_d, \sigma_d) = 0$ **return** 0
$c\mu^* \leftarrow\$ \mathbb{G}_2^3, c_1^* \leftarrow D_3, c_2^* \leftarrow m_b D_4$
$c\sigma^* \leftarrow\$ S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, c\mu^*, c^*))$
$b^* \leftarrow\$ \mathcal{A}(\mathsf{guess}, \mathsf{st}, c^* c\mu^*, c\sigma^*)$ **if** $b^* = b$ **return** 1 **return** 0

Figure 5.8: $\mathcal{D}_3$ that distinguishes between Game 3 and Game 4 in the CGS blindness proof

negligible.

We show that Game 3 and Game 4 are indistinguishable assuming the DDH assumption in $\mathbb{G}_1$. We provide a distinguishing algorithm $\mathcal{D}_3$ in Figure 5.8.

If $\mathcal{D}_3$ is input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 3. This is because, letting $\gamma = \log_g D_3$, then $D_4 = bpk_1^\gamma$, and so c^* is distributed identically to Game 3.

If $\mathcal{D}_3$ is not input a DDH tuple, all inputs to $\mathcal{A}$ are distributed identically to Game 4. This is because D_3, D_4 are now chosen independently and randomly.

Game 4 and Game 5 are indistinguishable assuming the DDH assumption in $\mathbb{G}_2$, by the exact same argument as that Game 3 and Game 4 are indistinguishable, as *cupk* is an ElGamal encryption of upk_b and c^* is an ElGamal encryption of m_b.

Therefore $|\Pr[P_3] - \Pr[P_4] \leq \epsilon_{\mathsf{DDH}}$, and $|\Pr[P_4] - \Pr[P_5] \leq \epsilon_{\mathsf{DDH}}$. Therefore $|\Pr[P_0] - \Pr[P_5]| \leq 3\epsilon_{\mathsf{DDH}} + \epsilon_{\mathsf{sdp}}$, and so $|\Pr[P_0] - 1/2| \leq 3\epsilon_{\mathsf{DDH}} + \epsilon_{\mathsf{sdp}}$. Therefore, assuming the DDH assumption and the Strong Derivation Privacy of the cm-NIZK, the advantage of any polynomial–time adversary in the blindness game with the CGS–cmNIZK construction is negligible

□

$\mathsf{SNDU}(uid, M_{\mathsf{in}})$

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$ $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, Q \leftarrow Q + 1,$ **if** $Q = k$ $uid^* \leftarrow uid$
  $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
  if $uid = uid^*$ $\mathbf{usk}[uid] \leftarrow \perp$ $\mathbf{upk}[uid] \leftarrow apk$
  else $(\mathbf{usk}[uid], \mathbf{upk}[uid]) \leftarrow_{\$} \mathsf{CGS.UKGen}(\mathsf{param})$
  return $(\mathbf{upk}[uid], \mathsf{cont})$
else continue from line 7 except **if** $uid = uid^*$ simulate π_{join}

$\mathsf{SIGN}(uid, m, bpk)$

if $uid = uid^*$
  if $\mathbf{dec}^{uid} \neq \mathsf{accept}$ **return** $\perp$
  Generate μ, c as usual using $\mathbf{upk}[uid]$
  $\sigma \leftarrow_{\$} S_2(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, \mu, c)), \mathsf{SL} \leftarrow \mathsf{SL} \cup \{(uid, m, bpk)\}$
  return (σ, μ)
else identical to non–frameability experiment

$\mathcal{A}'^{\mathsf{SIGN}_{\mathsf{auto}}}((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}}, apk)$

$Q \leftarrow 0, k \leftarrow_{\$} [q], \mathsf{param}_{\mathsf{auto}1} \leftarrow \mathsf{param}_{\mathsf{auto}}, \mathsf{param}_{\mathsf{auto}2} \leftarrow_{\$} \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$
$g \leftarrow_{\$} \mathbb{G}_1, \hat{g} \leftarrow_{\$} \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s, \tau_e) \leftarrow_{\$} \mathsf{SE}_1(1^\tau),$
$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$
$(ipk, isk) \leftarrow_{\$} \mathsf{CGS.IKGen}(\mathsf{param}), (opk, osk) \leftarrow_{\$} \mathsf{CGS.OKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(uid, c^*, c\mu^*, c\sigma^*, bpk^*, bsk^*) \leftarrow_{\$} \mathcal{A}^{\mathsf{SNDU}, \mathsf{SIGN}}(gpk, isk, osk)$
if $(bpk^*, bsk^*) \notin \mathcal{BK}$ **return** 0
$((\cdot, upk, \cdot, \Omega, \cdot, \cdot, \cdot, m), \cdot, \cdot) \leftarrow E_2(\sigma_{\mathsf{crs}}, \tau_e, (opk, bpk^*, ipk, c\mu^*, c^*), c\sigma^*)$
if $upk = apk$ **return** (m, Ω) **else return** $\perp$

Figure 5.9: $\mathcal{A}'$ which breaks the EUF-cma security of the automorphic signatures used, using $\mathcal{A}$ which breaks the non–frameability requirement of CGS–cmNIZK with probability ϵ

5.5.6 Non–frameability

Lemma 5.3. The CGS–cmNIZK construction presented in Section 5.4 satisfies **non–frameability** if the automorphic signature scheme is EUF-cma secure, the SPK is zero–knowledge, and the cm-NIZK is zero–knowledge and cm-SSE.

Proof. First we show that if there exists an adversary $\mathcal{A}$ that makes q queries to the SNDU oracle for distinct users, such that $\Pr[\mathbf{Exp}^{non-frame}_{\mathcal{A},\mathsf{CGS–cmNIZK}}(\tau) = 1] = \epsilon$, where ϵ is non-negligible, then we can can build an adversary $\mathcal{A}'$ that breaks the EUF-cma security of the automorphic signature scheme with non-negligible probability. We provide the detailed description of $\mathcal{A}'$ in Figure 5.9, and explain here how $\mathcal{A}'$ works.

First note that all inputs that $\mathcal{A}'$ provides to $\mathcal{A}$ are distributed identically to the non–frameability experiment. This is because SE_1 outputs a σ_{crs} that is identical to CRSSetup. The automorphic signature parameters $\mathsf{param}_{\mathsf{auto}}$ are distributed identically to the output of ASetup.

Simulating the SNDU oracles. When $uid \neq uid^*$, the SNDU oracle is identical to the non–frameability experiment. When $uid = uid^*$, $\mathbf{upk}[uid^*]$ is set to apk, which is distributed identically to the output of CGS.UKGen. The values $\mathbf{usk}[uid], upk'$ are set to $\perp$, but these are only used in the protocol when generating π_{join}. Instead, due to the zero–knowledge property of the SPK, π_{join} can be simulated using $\mathbf{upk}[uid]$.

Simulating the SIGN oracle. In the case of $uid \neq uid^*$, this is identical to the non–frameability experiment. When $uid = uid^*$, we first test $\mathbf{dec}^{uid} \neq \mathsf{accept}$ instead of whether the secret key is defined, because in our simulation the secret key will not be defined even for a completed join protocol. The μ, c are generated identically to CGS.Sign. The proof σ can be simulated due to the zero–knowledge property of the cm-NIZK proofs.

Reduction to EUF-cma security of Automorphic Signatures. If $\mathcal{A}$ is successful, then they output a valid blinded signature that opens to the upk of an honest user. We assume, with probability $1/q$, that $\mathcal{A}'$ guesses correctly and $uid = uid^*$.

With $1 - \mathsf{negl}$ probability, E_2 will either extract a valid witness such that $\mathsf{AVerify}_1(\Omega, \mathbf{upk}[uid^*], m) = 1$; or E_2 will extract a statement x' and transformation T such that $x = T_{\mathsf{inst}}(x')$, where x is the statement output, $T \in \mathcal{T}$ and $x' \in Q$.

In the first case $\mathcal{A}'$ wins because $\mathbf{upk}[uid^*] = apk$ and $\mathcal{A}'$ has never used their signing oracle. Therefore Ω is a valid forgery in the EUF-cma game.

In the second case, a proof must have been simulated during signing for a statement $x' = (opk, bpk^*, ipk, c\mu', c')$ such that $c\mu^*$ is a re-randomisation of $c\mu'$ and c^* is a re-randomisation of c'. Therefore, $c\mu'$ must be an encryption of $\mathbf{upk}[uid^*]$ and c' must be an encryption of m, and so (uid^*, m, bpk^*) was queried to the SIGN oracle. This is a contradiction given that $\mathcal{A}$ is successful. Therefore $\mathcal{A}'$ wins with the probability

$\mathsf{ADDU}(uid)$

Identical to the traceability experiment except $cred \leftarrow\!\$\, \mathsf{SIGN}_{\mathsf{auto}}(\mathbf{upk}[uid])$

$\mathsf{SNDI}(uid, M_{\mathsf{in}})$

Identical to the traceability experiment except $cred \leftarrow\!\$\, \mathsf{SIGN}_{\mathsf{auto}}(\mathbf{upk}[uid])$

$\mathsf{SIGN}(uid, m, bpk)$

Identical to the traceability experiment

$\mathcal{A}'^{\mathsf{SIGN}_{\mathsf{auto}}}((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}}, apk)$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow\!\$\, \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow \mathsf{param}_{\mathsf{auto}}$
$g \leftarrow\!\$\, \mathbb{G}_1, \hat{g} \leftarrow\!\$\, \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s, \tau_e) \leftarrow\!\$\, \mathsf{SE}_1(1^\tau)$
$\mathsf{param} \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}})$
$ipk \leftarrow apk, (opk, osk) \leftarrow\!\$\, \mathsf{CGS.OKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$
$(c, c\mu, c\sigma, bpk, bsk) \leftarrow\!\$\, \mathcal{A}^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SIGN}}(gpk, osk)$, **if** $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
$((\cdot, upk, cred, \cdot, \cdot, \cdot, \cdot, \cdot), \cdot, \cdot) \leftarrow E_2(\sigma_{\mathsf{crs}}, \tau_e, (opk, bpk, ipk, c\mu, c), c\sigma)$ **return** $(upk, cred)$

Figure 5.10: $\mathcal{A}'$ which breaks the EUF-cma security of automorphic signatures, using $\mathcal{A}$ which breaks the traceability of CGS–cmNIZK with probability ϵ

$(1 - \mathsf{negl})\epsilon/q$. □

5.5.7 Traceability

Lemma 5.4. The CGS–cmNIZK construction presented in Section 5.4 satisfies **traceability** if the automorphic signature scheme is EUF-cma secure, and the cm-NIZK is cm-SSE.

Proof. First, we show that if there exists an adversary $\mathcal{A}$ such that $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CGS–cmNIZK}}(\tau) = 1] = \epsilon$, where ϵ is non-negligible, then we can can build an adversary $\mathcal{A}'$ that breaks the EUF-cma security of the automorphic signature scheme with non-negligible probability. We provide the detailed description of $\mathcal{A}'$ in Figure 5.10, and explain here how $\mathcal{A}'$ works.

First note that all inputs that $\mathcal{A}'$ provides to $\mathcal{A}$ are distributed identically to the traceability experiment. This is because SE_1 outputs a σ_{crs} that is identical to CRSSetup. The inputs apk and $\mathsf{param}_{\mathsf{auto}}$ are the public key and public parameters of an automorphic signature scheme and so ipk and $\mathsf{param}_{\mathsf{auto}2}$ are distributed correctly.

Simulating the oracles. The ADDU and SNDI oracles are only different to the traceability experiment when *cred* is generated, because the issuing secret key is not available. As the signing oracle in the EUF-cma experiment outputs a valid signature under isk on the user public key, the resulting credential is distributed correctly. The SIGN oracle is identical to the traceability experiment, as all user secret keys are known.

Reduction to EUF-cma security of Automorphic Signatures. If $\mathcal{A}$ is successful, they output a valid blinded signature that does not open to the same upk as a corrupted or honest user.

We have not used S_1 to obtain simulations of the cm-NIZK proofs, therefore E_2 will extract a valid witness with $1-\mathsf{negl}$ probability. This ensures that $\mathsf{AVerify}_2(cred, ipk, upk) = 1$. As upk is not the public key of any corrupted or honest users, upk was not queried to $\mathsf{SIGN}_{\mathsf{auto}}$ by ADDU or SNDI. Therefore, *cred* is a valid forgery under apk, and $\mathcal{A}'$ is successful with probability $(1 - \mathsf{negl})\epsilon$. □

5.6 Concrete Instantiation and Efficiency

5.6.1 Signature Proofs of Knowledge

For transforming interactive into non-interactive zero-knowledge proofs we rely on the Fiat-Shamir heuristic that ensures security in the random oracle model.

5.6.2 Automorphic Signatures

An instantiation of an automorphic signature scheme that is EUF-cma secure, based on the *Asymmetric Double Hidden SDH* (ADHSDH) assumption, is given in [65]. It is easy to see that this scheme also satisfies the additional structural assumptions needed for our construction.

5.6.3 Controlled Malleable NIZKs

We now show that cm-NIZKs for the relation $\mathcal{R}$, and set of allowable transformations $\mathcal{T}$ used in our construction can be instantiated.

It is shown in Theorem 4.5 in [45] that cm-NIZKS for $(\mathcal{R}, \mathcal{T})$ can be instantiated if $(\mathcal{R}, \mathcal{T})$ are *CM-friendly* which they define fully in Section C.1.

More formally we define the relation R such that $((opk, bpk_1, bpk_2, ipk, c\mu, c)$, $(upk', upk, cred, \Omega, g_1^\alpha, g_1^\beta, g_2^\gamma, m) \in R$ if and only if:

$$e(g_1, \mu_1) = e(g_1^\alpha, \hat{g}), e(g_1, \mu_2) = e(g_1^\beta, \hat{g}), e(g_1, \mu_3) = e(g_1, upk)e(g_1^\alpha, opk)e(g_1^\beta, bpk_2);$$

$$\mathsf{AVerify}_1(\Omega, upk, m) = 1, \mathsf{AVerify}_2(cred, ipk, upk) = 1;$$

$$e(c_1, g_2) = e(g, g_2^\gamma), e(c_2, g_2) = e(m, g_2)e(bpk_1, g_2^\gamma);$$

$$e(upk', g_2) = e(g_1, upk).$$

We define the allowable set of transformation $\mathcal{T} = \{(r_{\mathsf{enc1}}, r_{\mathsf{enc2}}, r_{\mathsf{enc3}}) : r_{\mathsf{enc1}}, r_{\mathsf{enc2}}, r_{\mathsf{enc3}} \in \mathbb{Z}_p^*\}$, such that for $T = (r_{\mathsf{enc1}}, r_{\mathsf{enc2}}, r_{\mathsf{enc3}})$, the transformation is as follows:

$$T_{\mathsf{inst}}(opk, bpk, ipk, \mu, c) = (opk, bpk, ipk, (\mu_1 \hat{g}^{r_{\mathsf{enc1}}}, \mu_2 \hat{g}^{r_{\mathsf{enc2}}}, \mu_3 opk^{r_{\mathsf{enc1}}} bpk_2^{r_{\mathsf{enc2}}}),$$

$$(c_1 g^{r_{\mathsf{enc3}}}, c_2 bpk_1^{r_{\mathsf{enc3}}})),$$

$$T_{\mathsf{wit}}(upk', upk, cred, \Omega, g_1^\alpha, g_1^\beta, g_2^\gamma, m) = (upk', upk, cred, \Omega, g_1^\alpha g_1^{r_{\mathsf{enc1}}}, g_1^\beta g_1^{r_{\mathsf{enc2}}}, g_2^\gamma g_2^{r_{\mathsf{enc3}}}, m).$$

We now show that $(\mathcal{R}, \mathcal{T})$ is CM-Friendly which means six conditions are satisfied.

1. *Representable statements: any instance and witness of $\mathcal{R}$ can be represented as a set of group elements*

   Verification keys, messages and signatures of automorphic signatures are all group elements and so $ipk, upk, cred, \Omega, m$ are all group elements. The value $upk' = g_1^{usk}$ is also a group element.

   The values $opk, bpk_1, bpk_2, c\mu = (c\mu_1, c\mu_2, c\mu_3), c = (c_1, c_2)$ can all clearly be rep-

resented by group elements due to the ElGamal encryption used. The encryption randomness α, β, γ can be represented by $(g_1^{\alpha}, g_1^{\beta}, g_2^{\gamma})$.

2. *Representable transformations: any transformation in T can be represented as a set of group elements*

   Represent randomness $r_{\mathsf{enc1}}, r_{\mathsf{enc2}}, r_{\mathsf{enc3}}$ with $(g_1^{r_{\mathsf{enc1}}}, \hat{g}^{r_{\mathsf{enc1}}}, opk^{r_{\mathsf{enc1}}}, g_1^{r_{\mathsf{enc2}}}, \hat{g}^{r_{\mathsf{enc2}}}, bpk_2^{r_{\mathsf{enc2}}}, g_2^{r_{\mathsf{enc3}}}, g^{r_{\mathsf{enc3}}}, bpk_1^{r_{\mathsf{enc3}}})$.

3. *Provable statements: we can prove the statement $(x, w) \in \mathcal{R}$ (using the representation in condition 1 for x and w) using pairing product equations.*

   $\mathsf{AVerify}_1(\Omega, upk, m) = 1$, and $\mathsf{AVerify}_2(cred, ipk, upk) = 1$ can be written as a conjunction of pairing product equations over $upk, m, \Omega, cred, ipk$ due to the properties of automorphic signatures.

   All other equations are already in the form of pairing product equations.

4. *Provable transformations: we can prove the statement $T_{inst}(x) = x'$ for $T \in \mathcal{T}$ (using the representations for x and T in conditions 1 and 2) using a pairing product equation.*

   Given $x = (opk, bpk_1, bpk_2, ipk, (c\mu_1, c\mu_2, c\mu_3), (c_1, c_2))$,
   $x' = (opk', bpk_1', bpk_2', ipk', (c\mu_1', c\mu_2', c\mu_3'), (c_1', c_2'))$,
   $T = (R_1, R_1', R_1'', R_2, R_2', R_2'', R_3, R_3', R_3'')$, then $T_{\mathsf{inst}}(x) = x'$ and $T \in \mathcal{T}$ iff:

   $e(g_1, opk) = e(g_1, opk')$,$e(bpk_1, g_2) = e(bpk_1', g_2)$, $e(g_1, bpk_2) = e(g_1, bpk_2')$, $e(ipk, g_2) = e(ipk', g_2)$ and

   $e(c_2', g_2) = e(bpk_1, R_3)e(c_2, g_2)$, $e(c_1', g_2) = e(g, R_3)e(c_1, g_2)$, and

   $e(g_1, c\mu_3') = e(R_1, opk)e(R_2, bpk)e(g_1, c\mu_3)$,
   $e(g_1, c\mu_1') = e(R_1, \hat{g})e(g_1, c\mu_1)$, $e(g_1, c\mu_2') = e(R_2, \hat{g})e(g_1, c\mu_2)$.

5. *Transformable statements: for any $T \in \mathcal{T}$, there is a valid transformation $s(T)$ that takes the statement $(x, w) \in \mathcal{R}$ (phrased using pairing products as in condition 3) and produces the statement $(T_{inst}(x), T_{wit}(w)) \in \mathcal{R}$.*

   We transform the pairing product equations for an instance $(opk, bpk_1, bpk_2, ipk, (c\mu_1, c\mu_2, c\mu_3), (c_1, c_2))$ into mauled equations for an instance $(opk, bpk_1, bpk_2, ipk, (c\mu_1\hat{g}^{\alpha'}, c\mu_2\hat{g}^{\beta'}, c\mu_3 opk^{\alpha'} bpk_2^{\beta'}), (c_1 g^{\gamma'}, c_2 bpk_1^{\gamma'}))$ as follows:

   First of all we re-randomise $c\mu$.

- $\mathsf{Add}(eq_1 := e(g_1, \hat{g}^{\alpha'})^{-1} e(g_1^{\alpha'}, \hat{g}) = 1)$

  $\mathsf{Add}(eq_2 := e(g_1, \hat{g}^{\beta'})^{-1} e(g_1^{\beta'}, \hat{g}) = 1)$

  $\mathsf{Add}(eq_3 : e(g_1, opk^{\alpha'} bpk_2^{\beta'})^{-1} e(g_1^{\alpha'}, opk) e(g_1^{\beta'}, bpk_2) = 1)$

- $\mathsf{MergeEq}(eq_1,\ e(g_1, c\mu_1)^{-1} e(g_1^{\alpha}, \hat{g}) = 1\ )$ to create equation eq_4.

  $\mathsf{MergeEq}(eq_2,\ e(g_1, c\mu_2)^{-1} e(g_1^{\beta}, \hat{g}) = 1)$ to create equation eq_5.

  $\mathsf{MergeEq}(eq_3,\ e(g_1, c\mu_3)^{-1} e(g_1^{\alpha}, opk) e(g_1^{\beta}, bpk_2) e(g_1, upk) = 1\ )$ to create equation eq_6.

- $\mathsf{MergeVar}(c\mu_1, \hat{g}^{\alpha'}, c\hat{\mu}_1, \{g_1\})$, will create equation
  $e(g_1, c\mu_1 \hat{g}^{\alpha'}) e(g_1, c\hat{\mu}_1)^{-1} = 1.$
  $\mathsf{MergeEq}(e(g_1, c\mu_1 \hat{g}^{\alpha'}) e(g_1, c\hat{\mu}_1)^{-1} = 1, \mathsf{eq}_4\ )$ to create equation eq_7.

  $\mathsf{MergeVar}(c\mu_2, \hat{g}^{\beta'}, c\hat{\mu}_2, \{g_1\})$, will create equation
  $e(g_1, c\mu_2 \hat{g}^{\beta'},) e(g_1, c\hat{\mu}_2)^{-1} = 1.$
  $\mathsf{MergeEq}(e(g_1, c\mu_2 \hat{g}^{\beta'},) e(g_1, c\hat{\mu}_2)^{-1} = 1, \mathsf{eq}_5\ )$ to create equation eq_8.

  $\mathsf{MergeVar}(c\mu_3, opk^{\alpha'} bpk_2^{\beta'}, c\hat{\mu}_3, \{g_1\})$, will create equation
  $e(g_1, c\mu_3 opk^{\alpha'} bpk_2^{\beta'}) e(g_1, c\hat{\mu}_3)^{-1} = 1.$
  $\mathsf{MergeEq}(e(g_1, c\mu_3 opk^{\alpha'} bpk_2^{\beta'}) e(g_1, c\hat{\mu}_3)^{-1} = 1, \mathsf{eq}_6)$ to create equation eq_9.

- $\mathsf{MergeVar}(g_1^{\alpha'}, g_1^{\alpha}, g_1^{\tilde{\alpha}}, \{\hat{g}, opk\})$, will create equations $e(g_1^{\alpha'} g_1^{\alpha}, \hat{g})^{-1} e(g_1^{\tilde{\alpha}}, \hat{g}) = 1$ and $e(g_1^{\alpha'} g_1^{\alpha}, opk)^{-1} e(g_1^{\tilde{\alpha}}, opk) = 1.$

  $\mathsf{MergeEq}(e(g_1^{\alpha'} g_1^{\alpha}, \hat{g})^{-1} e(g_1^{\tilde{\alpha}}, \hat{g}) = 1, \mathsf{eq}_7\ ).$

  $\mathsf{MergeVar}(g_1^{\beta'}, g_1^{\beta}, g_1^{\tilde{\beta}}, \{\hat{g}, bpk_2\})$, will create equations $e(g_1^{\beta'} g_1^{\beta}, \hat{g})^{-1} e(g_1^{\tilde{\beta}}, \hat{g}) = 1$ and $e(g_1^{\beta'} g_1^{\beta}, bpk_2)^{-1} e(g_1^{\tilde{\beta}}, bpk_2) = 1.$

  $\mathsf{MergeEq}(e(g_1^{\beta'} g_1^{\beta}, \hat{g})^{-1} e(g_1^{\tilde{\beta}}, \hat{g}) = 1, \mathsf{eq}_8\ ).$

  $\mathsf{MergeEq}(e(g_1^{\alpha'} g_1^{\alpha}, opk)^{-1} e(g_1^{\tilde{\alpha}}, opk) = 1, e(g_1^{\beta'} g_1^{\beta}, bpk_2)^{-1} e(g_1^{\tilde{\beta}}, bpk_2) = 1)$ to create equation eq_{10} .

  $\mathsf{MergeEq}(\mathsf{eq}_9,\ \mathsf{eq}_{10}\ ).$

  Remove obsolete equations and variables with RemoveEq and RemoveVar.

We now re-randomise c.

- $\mathsf{Add}(eq_{11} := e(g^{\gamma'}, g_2)^{-1} e(g, g_2^{\gamma'}) = 1)$

  $\mathsf{Add}(eq_{12} := e(bpk_1^{\gamma'}, g_2)^{-1} e(bpk_1, g_2^{\gamma'}) = 1)$

- $\mathsf{MergeEq}(eq_{11}, e(c_1, g_2)^{-1} e(g, g_2^{\gamma}) = 1\ )$ to create equation eq_{13}.

  $\mathsf{MergeEq}(eq_{12},\ e(c_2, g_2)^{-1} e(bpk_1, g_2^{\gamma}) e(m, g_2) = 1\ )$ to create equation eq_{14}.

- $\mathsf{MergeVar}(c_1, g^{\gamma'}, \hat{c_1}, \{g_2\})$, will create equation $e(c_1 g^{\gamma'}, g_2)e(\hat{c_1}, g_2)^{-1} = 1$.
  $\mathsf{MergeEq}(e(c_1 g^{\gamma'}, g_2)e(\hat{c_1}, g_2)^{-1} = 1, \mathsf{eq}_{13})$ to create eq_{15}.
  $\mathsf{MergeVar}(c_2, bpk_1^{\gamma'}, \hat{c_2}, \{g_2\})$, will create equation $e(c_2 bpk_1^{\gamma'}, g_2)e(\hat{c_2}, g_2)^{-1} = 1$.
  $\mathsf{MergeEq}(e(c_2 bpk_1^{\gamma'}, g_2)e(\hat{c_2}, g_2)^{-1} = 1, \mathsf{eq}_{14})$ to create eq_{16}.
- $\mathsf{MergeVar}(g_2^{\gamma}, g_2^{\gamma'}, g_2^{\tilde{\gamma}}, \{bpk_1, g\})$, will create equations $e(g, g_2^{\gamma} g_2^{\gamma'})^{-1} e(g, g_2^{\tilde{\gamma}}) = 1$ and $e(bpk_1, g_2^{\gamma} g_2^{\gamma'})^{-1} e(bpk_1, g_2^{\tilde{\gamma}}) = 1$.
  $\mathsf{MergeEq}(e(g, g_2^{\gamma} g_2^{\gamma'})^{-1} e(g, g_2^{\tilde{\gamma}}) = 1, \mathsf{eq}_{15})$.
  $\mathsf{MergeEq}(e(bpk_1, g_2^{\gamma} g_2^{\gamma'})^{-1} e(bpk_1, g_2^{\tilde{\gamma}}) = 1, \mathsf{eq}_{16})$.

Finally remove obsolete equations and variables with RemoveEq and RemoveVar.

(6) *Transformable transformations: for any $T, T' \in \mathcal{T}$ there is a valid transformation $t(T)$ that takes the statement $T_{inst}(x') = x$ for $T \in \mathcal{T}$ (phrased using pairing products as in condition 4) and produces the statement $T'_{inst} \circ T_{inst}(x') = T'_{inst}(x)$ for $T' \circ T \in \mathcal{T}$ and that preserves the variables in x' (does not perform RemoveVar on variables in x').*

We transform the pairing product equations defined in (4) for proving knowledge of a transformation from instance $(opk, bpk, ipk, c\mu', c')$ to $(opk, bpk, ipk, c\mu, c)$, into mauled equations for proving knowledge of a transformation from instance $(opk, bpk, ipk, c\mu', c')$ to $(opk, bpk, ipk, (c\mu_1 \hat{g}^{\alpha}, c\mu_2 \hat{g}^{\beta}, c\mu_3 bpk_2^{\beta} opk^{\alpha}), (c_1 g^{\gamma}, c_2 bpk_1^{\gamma}))$. We can use the same strategy (and the same constant equations) as for transforming statements.

- $\mathsf{MergeVar}(c_1, g^{\gamma}, \hat{c_1}, \{g_2\})$, will create equation $e(c_1 g^{\gamma}, g_2)e(\hat{c_1}, g_2)^{-1} = 1$.
  $\mathsf{MergeEq}(e(c_1, g_2)^{-1} e(g, R_3) e(c'_1, g_2) = 1,\ e(c_1 g^{\gamma}, g_2)e(\hat{c_1}, g_2)^{-1} = 1)$.
  $\mathsf{MergeVar}(R_3, g_2^{\gamma}, \hat{R_3}, \{g, bpk_1\})$, will create equations $e(g, R_3 g_2^{\gamma})^{-1} e(g, \hat{R_3}) = 1$ and $e(bpk_1, R_3 g_2^{\gamma})^{-1} e(bpk_1, \hat{R_3}) = 1$.
  $\mathsf{MergeEq}(e(\hat{c_1}, g_2)^{-1} e(g, R_3) e(g, g_2^{\gamma}) e(c'_1, g_2),\ e(g, R_3 g_2^{\gamma})^{-1} e(g, \hat{R_3}) = 1)$.
  $\mathsf{MergeVar}(c_2, bpk_1^{\gamma}, \hat{c_2}, \{g_2\})$, will create equation $e(c_2 bpk_1^{\gamma}, g_2)e(\hat{c_2}, g_2)^{-1} = 1$.
  $\mathsf{MergeEq}(e(c_2, g_2)^{-1} e(bpk_1, R_3) e(c'_2, g_2) = 1,\ e(c_2 bpk_1^{\gamma}, g_2)e(\hat{c_2}, g_2)^{-1} = 1)$.
  $\mathsf{MergeEq}(e(\hat{c_2}, g_2)^{-1} e(bpk_1, R_3 g_2^{\gamma}) e(c'_2, g_2) = 1,\ e(bpk_1, R_3 g_2^{\gamma})^{-1} e(bpk_1, \hat{R_3}) = 1)$.
- $\mathsf{MergeVar}(c\mu_1, \hat{g}^{\alpha}, c\hat{\mu}_1, \{g_1\})$, will create equation $e(g_1, c\mu_1 \hat{g}^{\alpha}) e(g_1, c\hat{\mu}_1)^{-1} = 1$.
  $\mathsf{MergeEq}(e(g_1, c\mu_1)^{-1} e(R_1, \hat{g}) e(g_1, c\mu'_1) = 1,\ e(g_1, c\mu_1 \hat{g}^{\alpha})\ e(g_1, c\hat{\mu}_1)^{-1} = 1)$.

$\mathsf{MergeVar}(R_1, g_1^{\alpha}, \hat{R}_1, \{\hat{g}, opk\})$, will create equations $e(R_1 g_1^{\alpha}, \hat{g})^{-1} e(\hat{R}_1, \hat{g}) = 1$ and $e(R_1 g_1^{\alpha}, opk)^{-1} e(\hat{R}_1, opk) = 1$.

$\mathsf{MergeEq}(e(g_1, c\hat{\mu}_1)^{-1} e(R_1, \hat{g}) e(g_1^{\alpha}, \hat{g}) e(g_1, c\mu_1') = 1,\ e(R_1 g_1^{\alpha}, \hat{g})^{-1}\ e(\hat{R}_1, \hat{g}) = 1)$.

$\mathsf{MergeVar}(c\mu_2, \hat{g}^{\beta}, c\hat{\mu}_2, \{g_1\})$, will create equation $e(g_1, c\mu_2 \hat{g}^{\beta}) e(g_1, c\hat{\mu}_2)^{-1} = 1$.

$\mathsf{MergeEq}(e(g_1, c\mu_2)^{-1} e(R_2, \hat{g}) e(g_1, c\mu_2') = 1,\ e(g_1, c\mu_2 \hat{g}^{\beta})\ e(g_1, c\hat{\mu}_2)^{-1} = 1)$.

$\mathsf{MergeVar}(R_2, g_1^{\beta}, \hat{R}_2, \{\hat{g}, bpk_2\})$, will create equations $e(R_2 g_1^{\beta}, \hat{g})^{-1} e(\hat{R}_2, \hat{g}) = 1$ and $e(R_2 g_1^{\beta}, bpk_2)^{-1} e(\hat{R}_2, bpk_2) = 1$.

$\mathsf{MergeEq}(e(g_1, c\hat{\mu}_2)^{-1} e(R_2, \hat{g}) e(g_1^{\beta}, \hat{g}) e(g_1, c\mu_2') = 1,\ e(R_2 g_1^{\beta}, \hat{g})^{-1}\ e(\hat{R}_2, \hat{g}) = 1)$.

$\mathsf{MergeVar}(c\mu_3, opk^{\alpha} bpk_2^{\beta}, c\hat{\mu}_3, \{g_1\})$, will create equation
$e(g_1, c\mu_3 opk^{\alpha} bpk_2^{\beta}) e(g_1, c\hat{\mu}_3)^{-1} = 1$.

$\mathsf{MergeEq}(e(g_1, c\mu_3)^{-1} e(R_1, opk) e(R_2, bpk_2) e(g_1, c\mu_3') = 1,$
$e(g_1, c\mu_3 opk^{\alpha} bpk_2^{\beta}) e(g_1, c\hat{\mu}_3)^{-1} = 1)$.

$\mathsf{MergeEq}(e(g_1, c\hat{\mu}_3)^{-1} e(R_1, opk) e(g_1^{\alpha}, opk) e(R_2, bpk_2) e(g_1^{\beta}, bpk_2)\ e(g_1, c\mu_3') = 1,$
$e(R_1 g_1^{\alpha}, opk)^{-1} e(\hat{R}_1, opk) = 1\ )$.

$\mathsf{MergeEq}(e(g_1, c\hat{\mu}_3)^{-1} e(\hat{R}_1, opk) e(R_2, bpk_2) e(g_1^{\beta}, bpk_2) e(g_1, c\mu_3')$
$= 1, e(R_2 g_1^{\beta}, bpk_2)^{-1} e(\hat{R}_2, bpk_2) = 1\ )$.

Finally remove obsolete equations and variables with RemoveEq and RemoveVar, $(c\mu', c')$ will be unaffected by RemoveVar.

The instantiation in [45] makes use of Groth Sahai proofs [77] to build malleable *NI-WIPOKs*. The instantiation of Groth Sahai proofs based on the DLIN assumption and structure preserving signatures based on the DLIN assumption were chosen for their instantiation. However, for our instantiation as we are working in the type–3 pairing setting, we make use of the instantiation of Groth Sahai proofs based on the *SXDH* assumption, i.e. the DDH assumption holds in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$. We make use of a more recently introduced structure preserving signature scheme [6] with better efficiency also in the type–3 setting and based on the SXDH assumption.

5.6.4 Efficiency

We now provide the sizes of signatures for the CGS–cmNIZK construction in terms of the number of group elements. We denote the length required to represent k elements in $\mathbb{G}_i$ such that $i \in \{1, 2\}$ as $k\mathbb{G}_i$. A signature consists of a cm-NIZK which in turn

consists of a Groth Sahai NIWIPoK [77] instantiated under the SXDH assumption. The cost of variables and equation for such proofs is given in Table 3 of [77]. The statement proven by the NIWIPoK includes: the statement from our cm-NIZK, of size $10\mathbb{G}_1 + 9\mathbb{G}_2$ as well as two automorphic signatures of total size $5\mathbb{G}_1 + 5\mathbb{G}_2$ and parameters for this scheme $3\mathbb{G}_1 + 3\mathbb{G}_2$; the verification key of the SPS in the cm-NIZK instantiation of size $36\mathbb{G}_1 + 30\mathbb{G}_2$; a signature of the SPS in the cm-NIZK instantiation of size $14\mathbb{G}_1 + 14\mathbb{G}_2$; another transformed statement also of size $18\mathbb{G}_1 + 17\mathbb{G}_2$; and the transformation of size $4\mathbb{G}_1 + 5\mathbb{G}_2$. In total this gives $90\mathbb{G}_1 + 83\mathbb{G}_2$. Therefore the total cost of variables is $180\mathbb{G}_1 + 166\mathbb{G}_2$. The total number of pairing equations in the NIWIPoK includes: 12 equations from our cm-NIZK, 16 equations to verify the SPS, and 9 equations to ensure that the transformation is valid and transforms the two statements. In total this is 37 equations and the cost of these equations will be $148\mathbb{G}_1 + 148\mathbb{G}_2$. Therefore the total size of a signature is $328\mathbb{G}_1 + 314\mathbb{G}_2$.

5.7 Summary

In this chapter we propose commuting group signatures, which allow group signatures to be blinded whilst preserving the verifiability of signatures. We have formally defined the security for such a primitive, and given a provably secure construction in this model. This construction makes use of ElGamal encryption, automorphic signatures, and controlled malleable proof protocols. We then provide a concrete instantiation for these building blocks and analyse the efficiency of our construction.

Chapter 6

Convertible Group Signatures – Stronger Security and Preserved Verifiability

Contents

6.1 Introduction

In this chapter we strengthen the security of the CLS primitive introduced in Chapter 4, by removing the assumption that the entity that submits signatures to a converter is honest–but–curious. We strengthen the existing security model to take this into account, and provide a construction that provably satisfies this model using commuting group signatures.

6.1.1 Motivation and Background

In Chapter 4 a flexible variant of linkability was proposed for group signatures in the form of *convertible group signatures with selective linkability* (CLS). While all group signatures are fully unlinkable by default, certain subsets can be converted into a linked representation. The conversion is performed obliviously by a trusted *converter* that blindly transforms a batch of pseudonyms, mapping different pseudonyms stemming from the same user into the same one. To avoid the gradual reconstruction of all correlations, converted pseudonyms obtained through *different* queries remain unlinkable, i.e., conversions are strictly *non-transitive*.

However, this assumes that the party receiving fully unlinkable signatures (the *data lake*) and the one obtaining the converted ones (the *data processor*) to be the same entity, or at least belong to the same trust domain: the scheme only converts pseudonyms but not the actual signatures, i.e., the authenticity of data gets lost in the conversion process. Therefore a data processor must trust the data lake that converted data originates from actual user data. It also assumes this combined entity to be honest, as the input to the converter is considered to be well-formed, in the sense that the security guarantees only hold when "valid" pseudonyms are converted for which correct group signatures exist. As the converter receives *blinded* pseudonyms and no group signatures, this assumption is impossible to enforce other than by considering honest requests only.

6.1.2 Our Contribution

In this chapter we strengthen the concept of convertibly linkable group signatures (CLS+) to overcome these assumptions. We guarantee the desired security and privacy properties even when conversion is triggered by malicious entities. Further, we leverage the trusted converter to not only blindly transform the pseudonyms but also blindly re-authenticate the associated messages, thus preserving the authenticity of the data during the conversion process.

We start by extending the security and privacy definition given in Chapter 4 to this stronger setting. Our security model takes inspiration from the RCCA security notion [43], and grants the adversary the power to request conversions of arbitrary and blinded inputs,

while still preventing trivial replay attacks. We then propose a construction that provably satisfies the desired properties. The core building block for our construction is the primitive of *commuting group signatures* defined in Chapter 5.

Commuting group signatures naturally lend themselves to building convertible group signatures. When requesting a conversion, group signatures first get blinded, yet still allow verification by the converter. The converter then blindly opens the signatures and raises the blinded user identity to the power of r, which is chosen fresh for every conversion query, but used consistently within the query. The converter then blindly signs the converted pseudonym and message to attest that they originate from a valid query that contained blinded, yet verifiable, signatures. As we assume the converter is at most honest-but-curious, the authentication of converted signatures is carried over from that of the blinded signatures. During unblinding, the message and public key are unblinded, and the original output of the converter is included in the signature, along with a proof that unblinding has been performed correctly. We prove that our construction is secure, assuming the security of the commuting group signatures, as well as the DDH assumption.

6.2 Definition and Security Model for CLS+

In this section we recall the functionality and security model for CLS signatures given in Section 4.2 and extend them to our setting, which no longer assumes that conversion of pseudonymous signatures is triggered by *honest* verifiers only — we refer to these extended CLS signatures as CLS+. In fact, whereas CLS only converts pseudonyms but not the actual group signatures, our CLS+ scheme transforms pseudonyms while preserving the validity of the associated signatures. This requires several extensions to the algorithms and changes to the security model.

As in CLS, our CLS+ scheme assumes the following entities: an *issuer* $\mathcal{I}$, a set of users $\mathcal{U} = \{uid_i\}$, and a *converter* $\mathcal{C}$. In contrast to CLS, we split the verifier role into two parts: a *data lake*[1] $\mathcal{L}$ and *data processor* $\mathcal{P}$. This reflects our setting which allows a data processor to still verify the validity of the converted group signatures, whereas CLS assumed that unlinkable and converted signatures are both used by the same entity (or at

[1] We sometimes make the role of the data lake explicit for illustrative purposes, but this entity does not have any dedicated keys and any verifier can take this role.

least within the same trust domain).

The issuer $\mathcal{I}$ is the central entity that allows users to ***join*** the group. Once joined, a user can then ***sign*** on behalf of the group in a pseudonymous way. The data lake (or *any* verifier) can collect and verify these signatures with respect to the group's public key, but does not learn anything about the user that created the signature. While group signatures are fully unlinkable by default, they can be linked in a controlled, yet blind, manner. Such conversion can be requested by the data lake from the converter $\mathcal{C}$ that can blindly transform pseudonym-message-signature tuples into a consistent authenticated representation. The data lake blinds the signatures in the conversion request for a particular data processor $\mathcal{P}$, who can finally unblind and verify the converted signatures. Once unblinded, again *any* verifier can check the validity of the converted signature.

6.2.1 Syntax of CLS+

Our notation closely follows the definitional framework for CLS given in Section 4.2.1, but extends the blinding, conversion and unblinding procedure to not only transform pseudonyms and messages, but also signatures. Verification is extended to handle "standard" group signatures as well as converted signatures, allowing the data processor (or any verifier) to verify the transformed and linkable signatures.

Definition 6.1 (CLS+)**.** A convertibly linkable group signature scheme with preserved verifiability CLS+ consists of the following algorithms:

Setup and Key Generation. We model key generation per party, each generating their individual key pair. We refer to $(\mathsf{param}, ipk, cpk)$ as the group public key gpk and write $\mathcal{BK}$ to denote the public/private key space induced by CLS+.BKGen.

CLS+.Setup$(1^\tau) \to$ **param:** on input a security parameter 1^τ, outputs param, the public parameters for the scheme.

CLS+.IKGen(**param**) $\to (ipk, isk)$**:** performed by the issuer $\mathcal{I}$, outputs the issuer secret key isk, and the issuing public key ipk.

CLS+.CKGen(**param**) $\to (cpk, csk)$**:** performed by the converter $\mathcal{C}$, outputs the converter secret key csk, and the converter public key cpk.

CLS+.BKGen(param) $\rightarrow (bpk, bsk)$: performed by the data processor $\mathcal{P}$, outputs the blinding public key bpk and blinding private key bsk.

Join, Sign and Verify. We have a dedicated join procedure that a user has to complete with the issuer. All users that have successfully joined the group can then create pseudonymous signatures on behalf of the group. For ease of expression, we treat the pseudonym μ as a dedicated part of the signature.

Our construction requires that the user already specifies the data processor's key bpk when creating signatures, and thus we reflect this in the syntax. While this limits the flexibility of the data lake — it has to adhere to the choice of the user — it gives the users strong control over the usage of their data, as only they can choose who can unblind the converted (= linkable) signatures.

To handle our setting where converted signatures can also be verified, CLS+.Verify takes an extra input $type = \{\texttt{tier-1}, \texttt{tier-2}\}$ that indicates the type of signature: We denote standard, fully unlinkable signatures produced by CLS+.Sign as `tier-1` signatures, whereas converted ones (generated via processing a `tier-1` signature with CLS+.Blind – CLS+.Convert – CLS+.Unblind) are referred to as `tier-2` signatures.

$\langle$**CLS+.Join**(gpk), **CLS+.Issue**$(isk, gpk)\rangle$**:** a user uid joins the group by engaging in an interactive protocol with the Issuer $\mathcal{I}$. The user uid and Issuer $\mathcal{I}$ perform algorithms CLS+.Join and CLS+.Issue respectively. These are input a state and an incoming message respectively, and output an updated state, an outgoing message, and a decision, either cont, accept, or reject. The initial input to CLS+.Join is the group public key, gpk, whereas the initial input to CLS+.Issue is the issuer secret key, isk, and the group public key. If the user uid accepts, CLS+.Join has a private output of $\mathbf{gsk}[uid]$.

CLS+.Sign$(gpk, bpk, \mathbf{gsk}[uid], m) \rightarrow (\mu, \sigma)$**:** performed by the user uid with respect to a particular data processor with key bpk. On input the group public key gpk, the user's secret key $\mathbf{gsk}[uid]$, and a message m, outputs a pseudonym μ and signature σ.

CLS+.Verify$(type, gpk, bpk, m, \mu, \sigma) \rightarrow \{0, 1\}$**:** performed by any verifier. Outputs 1 if σ is a valid $\{\texttt{tier-1}, \texttt{tier-2}\}$-signature, on m for pseudonym μ under the group public

key gpk and data processor key bpk, and 0 otherwise.

Blind Conversion. Finally, we want our pseudonymous group signatures to be blindly convertible. Thus, there is a dedicated CLS+.Blind and CLS+.Unblind procedure for the data lake and data processor respectively, and a CLS+.Convert algorithm that requires the converter's secret key. The latter transforms the unlinkable pseudonyms in a consistent manner; i.e., outputting converted pseudonyms and signatures that are consistent whenever the input pseudonyms belong to the same user. As our scheme preserves the validity of the signatures, all algorithms from Section 4.2.1 are extended to also handle the signatures as in- and outputs.

CLS+.Blind$(gpk, bpk, (\mu, \sigma, m)) \to (c\mu, c\sigma, c)$**:** performed by the data lake with respect to a particular data processor bpk (and the converter specified in gpk); on input a pseudonym, signature and message, outputs a blinded pseudonym, signature and message.

CLS+.Convert$(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k) \to \{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k$**:** performed by the converter $\mathcal{C}$; on input k blinded pseudonym-signature-message tuples $(c\mu_1, c\sigma_1, c_1), \cdots,$ $(c\mu_k, c\sigma_k c_k)$, outputs converted tuples $(\overline{c\mu}_1, \overline{c\sigma_1}, \overline{c}_1), \cdots, (\overline{c\mu}_k, \overline{c\sigma_k}, \overline{c}_k)$.

CLS+.Unblind$(bsk, (\overline{c\mu}, \overline{c\sigma}, \overline{c})) \to (\overline{\mu}, \overline{\sigma}, m)$**:** performed by the data processor; on input a converted and blinded tuple, and the blinding secret key bsk, outputs an unblinded converted tuple $(\overline{\mu}, \overline{\sigma}, m)$.

6.2.2 Security Properties of CLS+

We want CLS+ schemes to enjoy (roughly) the same security and privacy properties as CLS schemes, without the assumption that conversion is triggered by honest parties only, and with the additional feature that converted data is also verifiable. We stress that our CLS+ schemes still rely on the converter being at most honest-but-curious though. We believe this to be an acceptable assumption in practice, as the converter is a central entity that can undergo more scrutiny than the multitude of verifiers and data lakes. To allow for a modular construction based on commuting group signatures, we no longer include the *join anonymity* requirement from Chapter 4. This is because commuting group signatures, as

in standard group signatures, can be opened to reveal the signer's identity and so do not provide join anonymity. As the converter is corrupted, this requirement only offers weak privacy guarantees anyway, and so we believe satisfying this requirement is less important than the positives of a simple modular construction.

CLS+ schemes must satisfy the following properties, where *anonymity, blindness* and *non–transitivity* capture the privacy-related properties and *non–frameability* and *traceability* formalise the desired unforgeability guarantees.

Anonymity: Group signatures which have not been linked through a conversion request should not leak any information about the signer's identity.

Non–transitivity: While conversion guarantees linkability *within* a batch of converted signatures, the data processor(s) should not be able to link the outputs of *different* convert queries.

Conversion Blindness: The converter should not learn anything about the input it receives nor the transformed outputs it computes.

Non–frameability: An adversary controlling the issuer and some corrupt users should not be able to impersonate other honest users i.e., create pseudonymous signatures that would be linked to a pseudonym of an honest user. Now we must also ensure that the adversary cannot forge converted signatures that link to those of honest users.

Traceability: An adversary controlling the issuer, converter and some users, should not be able to create more pseudonymous signatures that remain unlinkable in an (honest) conversion than they control corrupt users.

All notions require a number of changes to reflect the increased power of the data lake, while ensuring achievability of the properties. We explain these notions in detail in the following.

Oracles and State. The security notions we formalise in the following make use of a number of oracles which keep joint state, for example by keeping track of queries and the

set of corrupted parties. We present the detailed description of all oracles in Figure 6.1 and now provide an overview of them.

ADDU (join of honest user and honest issuer) Creates a new honest user for *uid* and internally runs a join protocol between the honest user and honest issuer. At the end, the honest user's secret key $\mathbf{gsk}[uid]$ is generated and from then on signing queries for *uid* will be allowed. This is identical to the CLS model.

SNDU (join of honest user and corrupt issuer) Creates a new honest user for *uid* and runs the join protocol on behalf of *uid* with the corrupt issuer. If the join session completes, the oracle will store the user's secret key $\mathbf{gsk}[uid]$. This is identical to the CLS model.

SNDI (join of corrupt user and honest issuer) Runs the join protocol on behalf of the honest issuer with corrupt users. For joins of honest users, the ADDU oracle must be used. This is identical to the CLS model.

SIGN This oracle returns signatures for honest users that have successfully joined (via ADDU or SNDU, depending on the game). This is identical to the CLS model, except the blinding public key is now input to CLS+.Sign, and so must be input to the oracle. Also, the blinding public key is now stored in SL, and the pseudonym and signatures no longer need to be stored as these could be re-randomised.

CONVERT Returns a set of converted signatures. The adversary must input the blinding secret key into the oracle, which is necessary for our privacy-related security notion, in anonymity to ensure that the adversary is not inputting a re-randomisation of the challenge signature. In the CLS+ model this now takes blinded inputs. The oracle detects whether the adversary has attempted to convert the challenge signature by checking whether inputs identify to the challenged user and unblind to the challenged message.

UNBLIND Blinds, converts and unblinds signatures, outputting the resulting `tier`-2 signatures. This is new to the CLS+ model. The oracle is necessary for the blindness requirement, because unblinding now outputs a `tier`-2 signature. This output could potentially leak information that allows other signatures to be unblinded. We blind, convert and unblind in this oracle because we are still targeting chosen plaintext level security.

All oracles have access to the following records maintained as global state:

HUL List of *uid*s of honest users, initially set to $\emptyset$. New honest users can be added by queries to the ADDU oracle (when the issuer is honest) or SNDU oracle (when the issuer is corrupt).

CUL List of corrupt users that have requested to join the group. Initially set to $\emptyset$, new corrupt users can be added through the SNDI oracle if the issuer is honest. If the issuer is corrupt, we do not keep track of corrupt users.

SL List of (uid, m, bpk) tuples requested from the SIGN oracle.

UBL List of $(\overline{\mu}, m, c, c\mu, c\sigma)$ tuples which are unblinded and converted pseudonyms/ messages along with the corresponding blinded signatures, stored by the CONVERT oracle.

Helper Algorithms. We use an adaptation of the helper algorithms from CLS for notational simplicity in our security games: Identify and UnLink. They now also take as input signatures, because they make use of CLS+.Convert. Identify determines whether a blinded signature belongs to a certain *uid*. We create a second signature for $\mathbf{gsk}[uid]$ and use the converter's secret key to test whether both signatures are linked. If so, Identify returns 1. This algorithm uses our second helper algorithm UnLink internally, which takes a list of (correctly formed) pseudonym-message-signatures pairs and returns 1 only if they are all unlinkable.

$\mathsf{Identify}(gpk, bpk, csk, bsk, uid, c, c\mu, c\sigma)$

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{CLS+.Sign}(gpk, bpk, \mathbf{gsk}[uid], 0), (c\mu', c\sigma', c') \leftarrow\!\!\$\ \mathsf{CLS+.Blind}(gpk, bpk, (\mu', \sigma', 0))$

if $\mathsf{UnLink}(gpk, csk, bpk, bsk, ((c\mu, c\sigma, c), (c\mu', c\sigma', c'))) = 0$ **return** 1 **else return** 0

$\mathsf{UnLink}(gpk, csk, bpk, bsk, ((c\mu_1, c\sigma_1, c_1), \cdots, (c\mu_k, c\sigma_k, c_k)))$

$\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k \leftarrow\!\!\$\ \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k)$

if $\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k = \perp$ **return** $\perp$

$\forall i \in [1, k] \quad (\overline{\mu}_i, \overline{\sigma}_i, m_i) \leftarrow\!\!\$\ \mathsf{CLS+.Unblind}(bsk, (\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i))$

if $\exists (i, j)$ with $i \neq j$ s.t. $\overline{\mu}_i = \overline{\mu}_j$ **return** 0 **else return** 1

ADDU(uid)

if $uid \in \mathsf{HUL} \cup \mathsf{CUL}$ **return** $\perp$
$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp$
$\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, \mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow gpk$
$\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk)$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CLS+.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, \perp)$
while $\mathbf{dec}^{uid} = \mathsf{cont}$
  $(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Join}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CLS+.Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Issue}})$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CLS+.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Join}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return accept

SIGN(uid, m, bpk)

if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$ **return** $\perp$
$(\mu, \sigma) \leftarrow\!\!\$\ \mathsf{CLS+.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)$
$\mathsf{SL} \leftarrow \mathsf{SL} \cup \{(uid, m, bpk)\}$
return (σ, μ)

SNDI(uid, M_{in})

if $uid \in \mathsf{HUL}$ **return** $\perp$
if $uid \notin \mathsf{CUL}$ $\mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{uid\}, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if undefined $\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk)$
$(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CLS+.Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{in}})$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

SNDU(uid, M_{in})

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$ $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}$
  $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if $\mathsf{st}^{uid}_{\mathsf{Join}}$ undefined $\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow gpk$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ \mathsf{CLS+.Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$

CONVERT($(c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk$)

if $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$, compute $\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k \leftarrow\!\!\$\ \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k)$
Parse permutation shuffling signatures in this run of CLS+.Convert as Π
$\forall i \in [1,k]$ $(\overline{\mu}_i, \overline{\sigma}_i, m_i) \leftarrow\!\!\$\ \mathsf{CLS+.Unblind}(bsk, (\overline{c\mu_{\Pi(i)}}, \overline{c\sigma_{\Pi(i)}}, \overline{c_{\Pi(i)}}))$, $\mathsf{UBL} \leftarrow \mathsf{UBL} \cup \{(\overline{\mu}_i, m_i, c_i, c\mu_i, c\sigma_i)\}$
if $\exists i \in [k]$ s.t $\mathsf{Identify}(uid^*_d, c_i, c\mu_i, c\sigma_i) = 1$ for $d \in \{0,1\}$ and $m_i = m^*$
  if $\exists j \in [k] \backslash \{i\}$ s.t. $\mathsf{Identify}(uid^*_d, c_j, c\mu_j, c\sigma_j) = 1$ for $d \in \{0,1\}$ **return** $\perp$
else return $(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k)$

UNBLIND($(\mu_1, \sigma_1, m_1), \ldots, (\mu_k, \sigma_k, m_k)$)

if $\exists i \in [k]$ s.t $\mathsf{CLS+.Verify}(\texttt{tier-1}, gpk, bpk, m_i, \mu_i, \sigma_i) = 0$ **return** $\perp$
$\forall i \in [k]$ $(c\mu_i, c\sigma_i, c_i) \leftarrow\!\!\$\ \mathsf{CLS+.Blind}(gpk, bpk, (\mu_i, \sigma_i, m_i); r_i)$
$\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k \leftarrow\!\!\$\ \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k; r')$
$\forall i \in [1,k]$ $(\overline{\mu}_i, \overline{\sigma}_i, m_i) \leftarrow\!\!\$\ \mathsf{CLS+.Unblind}(bsk, (\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i))$
return $(\{(\overline{\mu}_i, \overline{\sigma}_i, m_i)\}_k, \{(r_i)\}_k, r')$

Figure 6.1: Oracles used in our CLS+ security model

For simplicity we often omit the keys for the algorithms (as they are clear from the context). That is, we write $\mathsf{Identify}(uid, c, c\mu, c\sigma)$, which indicates whether the pseudonym $c\mu$ belongs to the user with identity uid or not. Likewise, we write $\mathsf{UnLink}((c_1, c\mu, c\sigma_1), \ldots, (c_k, c\mu_k, c\sigma_k))$ to test whether all pseudonyms are uncorrelated or not.

Correctness. CLS+ signatures should be correct and consistent when being produced by honest parties. We formulate correctness via three requirements, as in CLS:

- *Correctness of sign* guarantees that signatures formed using the CLS+.Sign algorithm with a user secret key generated honestly will verify correctly.

- *Correctness of conversion* guarantees that after blinding, converting and then unblinding correctly, the output will be correctly linked valid messages/ pseudonyms/ signatures.
- *Consistency* is a stronger variant of conversion-correctness and requires that the correlations of pseudonyms established through the conversion procedure must be consistent across queries. More precisely, if a conversion query reveals that two pseudonym μ_1 and μ_2 are linked, and another one that μ_2 and μ_3 are linked, then it must also hold that a conversion query for μ_1 and μ_3 returns linked pseudonyms.

The detailed definitions for correctness are given in Figure 6.2.

Definition 6.2 (Correctness)**.** A CLS+ scheme satisfies correctness if, for all adversaries $\mathcal{A}$, $\Pr[\mathbf{Exp}^{corr-sig}_{\mathcal{A},\mathsf{CLS+}}(\tau) = 1] = 0$, $\Pr[\mathbf{Exp}^{corr-conv}_{\mathcal{A},\mathsf{CLS+}}(\tau) = 1] \leq \mathsf{negl}(\tau)$, and, $\Pr[\mathbf{Exp}^{consist}_{\mathcal{A},\mathsf{CLS+}}(\tau) = 1] = 0$.

Again, for the *correctness of conversion*, the negligible chance that the adversary has of winning corresponds to the negligible chance that multiple user identifiers have the same secret key. The only aspect of correctness that has changed is that verify is now input converted signatures in correctness of conversion, and we check that converted signatures are valid.

Anonymity. This security requirement captures the desired anonymity properties when both the issuer and data lake are corrupt (but the converter is honest). Just as in CLS, we want signatures of honest users to be unlinkable and untraceable to a user's join session with the corrupt issuer. To model this property, we let the adversary output *uid*s of two honest users together with a message, and return a challenge (μ^*, σ^*) that is created either by user uid_0 or uid_1. For anonymity, the adversary should not be able to determine the user's identity with a better chance than by guessing.

The CLS notion assumed the data lake to be honest, which was modelled via a conversion oracle that took unblinded tuples as input, thus allowing the oracle to first check the validity of the input before then internally blinding and converting them. Here we allow the data lake to be malicious and therefore enable them to ask for the conversion of blinded tuples. In order to prevent trivial wins, we must be able to detect whether the adversary tries to convert the challenge signature. As signatures will be re-randomisable to allow

Experiment: $\mathbf{Exp}^{corr-sig}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS+.CKGen}(\mathsf{param})$
$(bpk, bsk) \leftarrow_\$ \mathsf{CLS+.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL} \leftarrow \emptyset$
$(uid, m) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU}}(gpk), \textbf{if } \mathbf{gsk}[uid] = \perp \quad \textbf{return } 0$
$(\mu, \sigma) \leftarrow_\$ \mathsf{CLS+.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)$
$\textbf{if } \mathsf{CLS+.Verify}(\texttt{tier-1}, gpk, bpk, m, \mu, \sigma) = 0 \quad \textbf{return } 1 \quad \textbf{else return } 0$

Experiment: $\mathbf{Exp}^{corr-conv}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS+.CKGen}(\mathsf{param}),$
$(bpk, bsk) \leftarrow_\$ \mathsf{CLS+.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL} \leftarrow \emptyset$
$((uid_1, m_0), ..., (uid_k, m_k)) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU}}(gpk)$
$\textbf{if } \exists i \in [1, k] \text{ st } \mathbf{gsk}[uid_i] = \perp \quad \textbf{return } 0$
$\forall i \in [1, k] \quad (\mu_i, \sigma_i) \leftarrow_\$ \mathsf{CLS+.Sign}(gpk, bpk, \mathbf{gsk}[uid_i], m_i)$
$\forall j \in [1, k] \quad (c\mu_j, c\sigma_j, c_j) \leftarrow_\$ \mathsf{CLS+.Blind}(gpk, bpk, (\mu_j, \sigma_j, m_j))$
$\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k \leftarrow_\$ \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k)$
$\forall j \in [1, k] \quad (\overline{\mu}_j, \overline{\sigma}_j, \overline{m}_j) \leftarrow_\$ \mathsf{CLS+.Unblind}((\overline{c\mu}_j, c\sigma_j, \overline{c}_j), bsk)$
$\textbf{if } \exists j \in [1, k] \text{ s.t } \mathsf{CLS+.Verify}(\texttt{tier-2}, gpk, bpk, \overline{m}_j, \overline{\mu}_j, \overline{\sigma}_j) = 0 \quad \textbf{return } 1$
$\textbf{if } \exists \text{ permutation } \Pi : [1, k] \to [1, k] \text{ s.t.}$
$\quad 1. \forall i \in [1, k] \quad m_{\Pi(i)} = m_i$
$\quad 2. \forall (i, j) \in [1, k] \text{ with } uid_i = uid_j \quad \overline{\mu}_{\Pi(i)} = \overline{\mu}_{\Pi(j)}$
$\quad 3. \forall (i, j) \in [1, k] \text{ with } uid_i \neq uid_j \quad \overline{\mu}_{\Pi(i)} \neq \overline{\mu}_{\Pi(j)}$
$\textbf{return } 0$
$\textbf{else return } 1$

Experiment: $\mathbf{Exp}^{consist}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS+.CKGen}(\mathsf{param}),$
$(bpk, bsk) \leftarrow_\$ \mathsf{CLS+.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$((c_0, c\mu_0, c\sigma_0), (c_1, c\mu_1, c\sigma_1), (c_2, c\mu_2, c\sigma_2)) \leftarrow_\$ \mathcal{A}(gpk, isk, csk, bsk)$
$\textbf{if } \mathsf{UnLink}((c_0, c\mu_0, c\sigma_0), (c_1, c\mu_1, c\sigma_1)) \neq 0 \text{ or}$
$\quad \mathsf{UnLink}((c_1, c\mu_1, c\sigma_1), (c_2, c\mu_2, c\sigma_2)) \neq 0 \quad \textbf{return } 0$
$\textbf{if } \mathsf{UnLink}((c_0, c\mu_0, c\sigma_0), (c_2, c\mu_2, c\sigma_2)) = 1 \quad \textbf{return } 1$
$\textbf{else return } 0$

Figure 6.2: Security games for correctness of CLS+

for non–transitivity, we opt for an RCCA-style of definition and check whether any of the blinded signature-tuples would identify to either of the challenge users and the challenge message. To do so, we also require the adversary to input the blinding *secret* key into the oracle. We stress that this key is merely used to check whether the internally unblinded inputs can be traced to a challenge user, but there are no checks that enforce that the inputs are actually well-formed.

Definition 6.3 (CLS+ Anonymity)**.** A CLS+ scheme satisfies anonymity if for all polynomial–time adversaries $\mathcal{A}$ the following advantage is negligible in τ :

$$\left|\Pr[\mathbf{Exp}^{anon-0}_{\mathcal{A},\mathsf{CLS}+}(\tau)=1]-\Pr[\mathbf{Exp}^{anon-1}_{\mathcal{A},\mathsf{CLS}+}(\tau)=1]\right|.$$

Experiment: $\mathbf{Exp}^{anon-b}_{\mathcal{A},\mathsf{CLS}+}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CLS}+.\mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow\!\!\$\ \mathsf{CLS}+.\mathsf{IKGen}(\mathsf{param}), (cpk, csk) \leftarrow\!\!\$\ \mathsf{CLS}+.\mathsf{CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk), (uid_0^*, uid_1^*, m^*, bpk^*, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVERT}}(\mathsf{choose}, gpk, isk)$

if $uid_0^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0^*] = \perp$ or $uid_1^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_1^*] = \perp$ **return** 0

$(\mu^*, \sigma^*) \leftarrow\!\!\$\ \mathsf{CLS}+.\mathsf{Sign}(gpk, bpk^*, \mathbf{gsk}[uid_b^*], m^*)$

$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVERT}}(\mathsf{guess}, \mathsf{st}, \mu^*, \sigma^*)$ **return** b^*

Non–transitivity. The second privacy-related property of CLS is the strict non-transitivity of conversions. This must now hold when not only the issuer, but also the data lake and processor can be fully corrupt. Non-transitivity ensures that the outputs of separate convert queries cannot be linked together across multiple queries. Otherwise, data processor(s) could gradually re-recover the linkability among all signatures.

The definition uses a simulation-based approach, requiring the indistinguishability of an ideal and a real world. In the real world, convert queries are handled normally through the CONVERT oracle defined in Figure 6.1. Whereas in the ideal world, the converted pseudonyms are produced by a simulator SIM through the CONVSIM oracle which we define here. For honest users, the simulator will only learn which of the messages belong together, but does not obtain any information about the underlying user identity. This naturally enforces the desired non-transitivity.

In contrast to the original CLS model, we now allow the data lake to trigger conversions on *blinded* inputs. Similarly to the anonymity game, the adversary must also input the blinding (secret) key with each query. Here the key is used to internally unblind the inputs

and determine the correlation among the signatures. We stress that this is not an artefact of our concrete instantiation, but rather a necessity to obtain a security definition that is realizable, as the CONVSIM oracle is still expected to provide consistently transformed outputs *within* a query. Further, the ideal CONVSIM oracle first internally runs the real conversion algorithm and aborts if it fails. This is again necessary to avoid trivial wins where the adversary might input malformed tuples which the simulator never gets and thus cannot verify.

Definition 6.4 (CLS+ Non–transitivity)**.** A CLS+ scheme satisfies non-transitivity if for all polynomial–time adversaries $\mathcal{A}$ there exists an efficient simulator SIM such that the following advantage is negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{nontrans-0}_{\mathcal{A},\mathsf{CLS+}}(\tau)=1]-\Pr[\mathbf{Exp}^{nontrans-1}_{\mathcal{A},\mathsf{CLS+}}(\tau)=1]\right|.$$

Experiment: $\mathbf{Exp}^{nontrans-b}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS+.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk)$

$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU,SIGN,CONVX}}(\mathsf{guess}, gpk, isk)$ **return** b^*

where the oracle CONVX works as follows:

if $b=0$ (real world) **then** CONVX is the standard CONVERT oracle

if $b=1$ (ideal world) **then** CONVX is the simulated CONVSIM oracle

$\mathsf{CONVSIM}((c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk)$

if $(bpk, bsk) \notin \mathcal{BK}$ or $\mathsf{CLS+.Convert}(gpk, bpk, csk, (c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k)) = \perp$ **return** $\perp$

$\mathsf{CL} \leftarrow \emptyset, \forall i \in [1,k]$

if $\exists uid \in \mathsf{HUL}$ s.t $\mathsf{Identify}(bpk, bsk, uid, c_i, c\mu_i, c\sigma_i) = 1$

if L_{uid} does not exist, create $\mathsf{L}_{uid} \leftarrow \{c_i\}$ **else** set $\mathsf{L}_{uid} \leftarrow \mathsf{L}_{uid} \cup \{c_i\}$

else $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(c_i, c\sigma_i, c\mu_i)\}$

$(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{i=1,\ldots k'}) \leftarrow_\$ \mathsf{CLS+.Convert}(gpk, bpk, csk, \mathsf{CL})$ for $k' \leftarrow |\mathsf{CL}|$

Let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters

$\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{i=k'+1,\ldots k} \leftarrow_\$ \mathsf{SIM}(gpk, bpk, csk, \mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}})$

Let $\{(\overline{c\mu}'_i, \overline{c\sigma}'_i, \overline{c}'_i)\}_{i=1,\ldots k}$ be a random permutation of $\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{i=1,\ldots k}$

return $(\{(\overline{c\mu}'_i, \overline{c\sigma}'_i, \overline{c}'_i)\}_{i=1,\ldots k})$

Conversion Blindness *(Corrupt Issuer and Converter).* As in CLS, a crucial property of our signatures is that they can be converted obliviously, i.e., the converter learns nothing about what it converts. However now the converter receives and outputs signatures, which must also be converted obliviously.

Conversion blindness should hold if both the issuer and converter are corrupt, but the data lake is honest. We formalise this property in a classic indistinguishability style: the adversary outputs two tuples of pseudonym-message-signature tuples and receives a blinded version of either of them.

In the original Conversion Blindness requirement for CLS no oracles are required because blinding is a public-key operation. However, in the CLS+ setting, CLS+.Unblind now outputs a `tier`-2 signature. We therefore must ensure that this signature does not leak anything that allows for the unblinding of other converted signatures. We therefore provide the adversary with access to a CLS+.Unblind oracle that blinds, converts and unblinds signatures. We stress that this requirement only provides chosen–plaintext level security as in CLS, because CLS+.Unblind both blinds and unblinds signatures.

Definition 6.5 (CLS+ Conversion Blindness)**.** A CLS+ scheme satisfies conversion blindness if for all polynomial–time adversaries $\mathcal{A}$ the following advantage is negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{blind-conv-0}_{\mathcal{A},\mathsf{CLS+}}(\tau)=1]-\Pr[\mathbf{Exp}^{blind-conv-1}_{\mathcal{A},\mathsf{CLS+}}(\tau)=1]\right|.$$

Experiment: $\mathbf{Exp}^{blind-conv-b}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow\!\!\$\ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow\!\!\$\ \mathsf{CLS+.CKGen}(\mathsf{param})$

$(bpk, bsk) \leftarrow\!\!\$\ \mathsf{CLS+.BKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, cpk)$

$(\mathsf{st}, (\mu_0, \sigma_0, m_0), (\mu_1, \sigma_1, m_1)) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{UNBLIND}}(\mathsf{choose}, gpk, bpk, isk, csk)$

if $\exists d \in \{0,1\}$ s.t $\mathsf{CLS+.Verify}(\texttt{tier-1}, gpk, bpk, m_d, \mu_d, \sigma_d) = 0$ **return** 0

$(c\mu^*, c\sigma^*, c^*) \leftarrow\!\!\$\ \mathsf{CLS+.Blind}(gpk, bpk, (\mu_b, \sigma_b, m_b))$

$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{UNBLIND}}(\mathsf{guess}, \mathsf{st}, c\mu^*, c\sigma^*, c^*)$ **return** b^*

Traceability of `tier`-2 signatures. This security requirement formalises the unforgeability properties when the issuer is honest but the converter, data lake and some users are corrupt. To lift the CLS traceability notion to the setting where conversion can be

triggered by malicious parties, we let the adversary output a list of *blinded* signatures. As we still assume the converter to be corrupt in an honest-but-curious form, the signatures are then *honestly* converted and unblinded by the challenger. The adversary wins if this conversion leads to more unlinkable signatures than the number of users it has corrupted plus the number of signatures obtained through signing queries for distinct users. Doing the unblinding within the game is necessary to check whether the adversary has indeed produced a valid forgery.

Note that this notion only captures traceability of `tier`-2 signatures, and CLS+ schemes must additionally satisfy the traceability of `tier`-1 signatures as defined in Section 4.2.

Definition 6.6 (CLS+ Traceability)**.** A CLS+ scheme satisfies `tier`-2 traceability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CLS+}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathbf{Exp}^{trace}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow\!\!\$\ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow\!\!\$\ \mathsf{CLS+.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk)$

$((c_1, c\mu_1, c\sigma_1), ..., (c_k, c\mu_k, c\sigma_k), bpk, bsk) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{ADDU,SNDI,SIGN}}(gpk, csk)$

if $(bpk, bsk) \notin \mathcal{BK}$ **return** 0

$\forall i \in [k] \quad (\overline{\mu}_i, \overline{\sigma}_i, m_i) \leftarrow\!\!\$\ \mathsf{CLS+.Unblind}(bsk, \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k))$

$L \leftarrow 0, \forall uid \in \mathsf{HUL}$ **if** $\exists i \in [k]$ s.t $(uid, m_i, bpk) \in \mathsf{SL} \quad L \leftarrow L + 1$

return 1 if all of the following conditions are satisfied:

$k > |\mathsf{CUL}| + L$ and $\mathsf{UnLink}((c\mu_1, c\sigma_1, c_1), ..., (c\mu_k, c\sigma_k, c_k)) = 1$ and

$\forall i \in [k] \quad \mathsf{CLS+.Verify}(\texttt{tier-2}, gpk, bpk, m_i, \overline{\mu}_i, \overline{\sigma}_i) = 1$

Traceability of `tier`-1 Signatures. For completeness we now present the original CLS traceability requirement for `tier`-1 signatures in the context of our new setting.

Definition 6.7 (CLS+ `tier`-1 Traceability)**.** A CLS+ scheme satisfies `tier`-1 traceability if for all polynomial–time adversaries $\mathcal{A}$, the advantage $\Pr[\mathbf{Exp}^{trace-tier1}_{\mathcal{A},\mathsf{CLS+}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathbf{Exp}^{trace-tier1}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow_\$ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow_\$ \mathsf{CLS+.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk)$

$((m_1, \mu_1, \sigma_1), ..., (m_k, \mu_k, \sigma_k), bpk) \leftarrow_\$ \mathcal{A}^{\mathsf{ADDU,SNDI,SIGN}}(gpk, csk)$

$\forall i \in [k] \quad (c\mu_i, c\sigma_i, c_i) \leftarrow_\$ \mathsf{CLS+.Blind}(gpk, bpk, (\mu_i, \sigma_i, m_i))$

$L \leftarrow 0, \forall uid \in \mathsf{HUL} \quad \textbf{if } \exists i \in [k] \text{ s.t } (uid, m_i, bpk) \in \mathsf{SL} \quad L \leftarrow L + 1$

return 1 if all of the following conditions are satisfied:

$k > |\mathsf{CUL}| + L$ and $\mathsf{UnLink}((c\mu_1, c\sigma_1, c_1), ..., (c\mu_k, c\sigma_k, c_k)) = 1$ and

$\forall i \in [k] \quad \mathsf{CLS+.Verify}(\texttt{tier-1}, gpk, bpk, m_i, \mu_i, \sigma_i) = 1$

Non–frameability. This notion captures the desired unforgeability properties when the issuer is corrupt and requires that an adversary should not be able to impersonate an honest user. Impersonation here means that the adversary outputs two converted signatures that link to an honest user, but that user never signed one of the corresponding messages. We use the Identify algorithm to check whether the signatures stem from an honest user and then look up whether at least one of the messages is *not* contained in the list of signed messages by that user.

As we defined Identify to work on blinded signatures, we first retrieve $(\overline{\mu_b}, m_b, c_b, c\mu_b, c\sigma_b)$ from the list of queries of the convert oracle (for $(\overline{\mu_b}, m_b)$). If such tuples cannot be found, the adversary also wins.

In contrast to traceability, it suffices to define non-frameability for `tier-2` signatures only: `tier-1` signatures are entirely unlinkable, and thus the notion of framing attacks only matters for converted signatures. Also, unlike in traceability, the adversary does *not* get the converter secret key here but instead outputs converted signatures directly (whereas traceability gave the adversary the key, but converted the signatures honestly).

Definition 6.8 (CLS+ Non–frameability)**.** A CLS+ scheme satisfies non–frameability if for all polynomial–time adversaries $\mathcal{A}$ the advantage $\Pr[\mathbf{Exp}^{nonframe}_{\mathcal{A},\mathsf{CLS+}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathbf{Exp}^{nonframe}_{\mathcal{A},\mathsf{CLS+}}(\tau)$

$\mathsf{param} \leftarrow\!\!\$\ \mathsf{CLS+.Setup}(1^\lambda), (ipk, isk) \leftarrow\!\!\$\ \mathsf{CLS+.IKGen}(\mathsf{param}), (cpk, csk) \leftarrow\!\!\$\ \mathsf{CLS+.CKGen}(\mathsf{param})$

$gpk \leftarrow (\mathsf{param}, ipk, cpk)$

$((m_0, \overline{\mu}_0, \overline{\sigma_0}), (m_1, \overline{\mu}_1, \overline{\sigma_1}), bpk, bsk) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU,SIGN,CONVERT}}(gpk, isk)$

if $(bpk, bsk) \notin \mathcal{BK}$ or $\exists b \in \{0,1\}$ s.t $\mathsf{CLS+.Verify}(\texttt{tier-2}, gpk, bpk, m_b, \overline{\mu}_b, \overline{\sigma}_b) = 0$ **return** 0

$\forall b \in \{0,1\}$ **if** $(\overline{\mu_b}, m_b, c_b, c\mu_b, c\sigma_b) \notin \mathsf{UBL}$ **return** 1

return 1 if all of the following conditions are satisfied:

$\overline{\mu}_0 = \overline{\mu}_1$ and $\exists uid \in \mathsf{HUL}$ s.t $\mathsf{Identify}(uid, c_0, c\mu_0, c\sigma_0) = 1$ and

$(uid, m_0, bpk) \notin \mathsf{SL}$, or $(uid, m_1, bpk) \notin \mathsf{SL}$

6.3 Our CLS+ Construction

We finally present our construction of a convertible group signature scheme. It uses a commuting group signature, as defined in Chapter 5, as a core building block provided it satisfies some additional structural assumptions, and also leverages a standard digital signature scheme SIG, and a signature proof of knowledge SPK.

High–level Idea. The issuer in our CLS+ scheme simply runs the same algorithms as in the CGS scheme, whereas the converter takes over the role of the opener in CGS and also creates a key pair for a standard signature scheme. When requesting blinded conversions, the group signatures are blinded and re-randomised using the algorithms from the CGS scheme. The converter blindly opens them and raises the blinded user keys to the power of r, which is chosen afresh in every convert query, but used consistently within the query. The converter then blindly signs the converted pseudonym and message. During unblinding, the converted pseudonyms and messages are retrieved. The final converted signature is the blindly signed tuple from the converter, along with a proof that unblinding has been done correctly, which can both be publicly verified.

To ensure that the converter only blindly signs messages that were correctly authenticated via a group signature, we use the property that blinded signatures from the CGS scheme can also be verified. Given that the converter is assumed to be at most honest-but-curious, this transmits the authentication guarantees from the original group signatures to the converted ones.

Additional Structural Assumptions of CGS. We need some additional assumptions from the underlying CGS scheme. We will show in Section 6.5 that these assumptions are satisfied by our concrete instantiation given in Chapter 5. Firstly, we require that given a bilinear group, $usk \in \mathbb{Z}_p^*, upk = g_2^{usk}$, and the output of CGS.OpenBlind consists of elements in $\mathbb{G}_2$. We also require that:

$$e(g_1, \mathsf{CGS.UnblindUser}(bsk, cupk))^r = \mathsf{CGS.UnblindUser}(bsk, e(g_1, cupk)^r) \text{ and}$$

$$\mathsf{CGS.BlindUser}(bpk, upk))^r = \mathsf{CGS.BlindUser}(bpk, upk^r).$$

This essentially means that unblinding of user public keys output by CGS.OpenBlind is homomorphic. This allows the converter to ensure non-transitivity by re-randomising the pseudonym output, whilst ensuring that pseudonyms are consistently linked per convert query.

We also require the CGS scheme to satisfy an additional property we define as *extractability*. Roughly, this allows upk' to be extracted from standard and blinded group signatures opening to upk, such that $e(upk', g_2) = e(g_1, upk)$, provided that the signature does not open to a user public key input to a simulator that simulates signatures.

We now present the extractability requirement for commuting group signatures.

Firstly, there exists simulators S_1, S_2, S_3 such that for all polynomial–time adversaries $\mathcal{A}$ the advantage $\left|\Pr[\mathbf{Exp}^{sim-0}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1] - \Pr[\mathbf{Exp}^{sim-1}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]\right|$ is negligible in τ.

Experiment: $\mathsf{Exp}^{\text{sim-b}}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$\ \mathcal{G}(1^\tau)$, **if** $b = 0$ $\quad \mathsf{param} \leftarrow\!\!\$\ \mathsf{CGS.Setup}(1^\tau)$

if $b = 1$ $\quad (\mathsf{param}, \tau_s) \leftarrow\!\!\$\ S_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$

$(isk, ipk) \leftarrow\!\!\$\ \mathsf{CGS.IKGen}(\mathsf{param}), (osk, opk) \leftarrow\!\!\$\ \mathsf{CGS.OKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$

$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDUX},\mathsf{SIGNX}}(gpk, isk, osk)$

where the oracles SNDUX, SIGNX work as follows:

if $b = 0$ (real world) **then** SNDUX, SIGNX are the SNDU, SIGN oracles

if $b = 1$ (ideal world) **then** SNDUX, SIGNX are the simulated SNDUSIM, SIGNSIM oracles

return b^*

SNDUSIM(uid, M_{in})

if $uid \in \mathsf{CUL}$ **return** $\perp$

if $uid \notin \mathsf{HUL}$ $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$

$\mathbf{upk}[uid] \leftarrow\!\!\$\ \mathbb{G}_2$

if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$, **if** $\mathsf{st}^{uid}_{\mathsf{Join}}$ undefined $\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk, \perp, \mathbf{upk}[uid])$

$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ S_2(\mathsf{param}, \mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}}, \tau_s)$, **return** $(M_{\mathsf{Out}}, \mathbf{dec}^{uid})$

SIGNSIM(uid, m, bpk)

if $uid \notin \mathsf{HUL}$ or $\mathbf{dec}^{uid} \neq \mathsf{accept}$ **return** $\perp$

$(\mu, \sigma) \leftarrow\!\!\$\ S_3(\mathsf{param}, gpk, bpk, \mathbf{upk}[uid], m, \tau_s)$ **return** (σ, μ)

Secondly, there exists SE_1, E_2, such that $(\mathsf{param}, \tau_s, \cdot)$ output by SE_1 is distributed identically to the output of S_1, and for all polynomial–time adversaries $\mathcal{A}$ the advantage $\Pr[\mathbf{Exp}^{ext}_{\mathcal{A},\mathsf{CGS}}(\tau) = 1]$ is negligible in τ.

Experiment: $\mathsf{Exp}^{\mathrm{ext}}_{\mathcal{A},\mathsf{CGS}}(\tau)$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow\!\!\$\ \mathcal{G}(1^\tau)$

$(\mathsf{param}, \tau_s, \tau_e) \leftarrow\!\!\$\ SE_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$

$(isk, ipk) \leftarrow\!\!\$\ \mathsf{CGS.IKGen}(\mathsf{param}), (osk, opk) \leftarrow\!\!\$\ \mathsf{OKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, opk)$

$(c, c\mu, c\sigma, bpk, bsk) \leftarrow\!\!\$\ \mathcal{A}^{S_2(\mathsf{param},\cdot,\cdot,\tau_s), S_3(\mathsf{param},\cdot,\cdot,\cdot,\tau_S)}(gpk, isk, osk)$

if $\mathsf{CGS.BlindVerify}(gpk, bpk, c, c\mu, c\sigma) = 0$ or $(bpk, bsk) \notin \mathcal{BK}$ **return** 0

$upk \leftarrow \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, osk, c, c\mu, c\sigma))$

if upk queried to S_3 for bpk, opk, ipk **return** 0

$upk' \leftarrow E_2(gpk, bpk, isk, osk, bsk, c, c\mu, c\sigma, \tau_e)$

if $upk' = \perp$ or $e(upk', g_2) \neq e(g_1, upk)$ **return** 1 **else return** 0

6.3.1 Detailed Description of CLS–CGS

We now give a detailed description of the CLS–CGS construction. Only commuting group signatures that satisfy the additional structural assumptions can be used to build this construction.

Setup and Key Generation. The CLS+.Setup , CLS+.IKGen, CLS+.CKGen and CLS+.BKGen algorithms are identical to the CGS.Setup, CGS.IKGen, CGS.OKGen and CGS.BKGen algorithms for CGS respectively, except that the converter's key also includes a signing and verification key of a digital signature scheme.

$\underline{\mathsf{CLS+.Setup}(1^\tau)}$

$\mathsf{param}_{\mathsf{cgs}} \leftarrow\!\!\$\ \mathsf{CGS.Setup}(1^\tau)$

$\mathsf{param}_{\mathsf{sig}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}(1^\tau)$

return $(\mathsf{param}_{\mathsf{cgs}}, \mathsf{param}_{\mathsf{sig}})$

$\underline{\mathsf{CLS+.IKGen}(\mathsf{param})}$

return $\mathsf{CGS.IKGen}(\mathsf{param}_{\mathsf{cgs}})$

$\underline{\mathsf{CLS+.BKGen}(\mathsf{param})}$

return $\mathsf{CGS.BKGen}(\mathsf{param}_{\mathsf{cgs}})$

$\underline{\mathsf{CLS+.CKGen}(\mathsf{param})}$

$(cpk_1, csk_1) \leftarrow\!\!\$\ \mathsf{CGS.OKGen}(\mathsf{param}_{\mathsf{cgs}}), (cpk_2, csk_2) \leftarrow\!\!\$\ \mathsf{SIG.KeyGen}(1^\tau)$

return $((cpk_1, cpk_2), (csk_1, csk_2))$

Join. The join protocol between a user and the issuer is identical to the join protocol of our CGS scheme, except the (upk, usk) are generated in CLS+.Join. The detailed protocol of $\langle \mathsf{CLS+.Join}(gpk), \mathsf{CLS+.Issue}(isk, gpk)\rangle$ is given in Figure 6.3.

$\mathcal{U}.\mathsf{CLS+.Join}(gpk)$	$\rightleftharpoons$	$\mathcal{I}.\mathsf{CLS+.Issue}(isk, gpk)$
$(upk, usk) \leftarrow\!\!\$\ \mathsf{CGS.UKGen}(\mathsf{param}_{\mathsf{cgs}})$		
	$\xrightarrow{upk}$	
$\langle \mathsf{CGS.Join}(gpk, usk, upk), \mathsf{CGS.Issue}(gpk, isk, upk)\rangle$		

Figure 6.3: Join protocol of our CLS–CGS construction

Sign and Verification of `tier`-1 signatures. Our signatures are identical to those in the CGS scheme, and therefore signing and verification are the same. In more detail, CLS+.Sign and CLS+.Verify are defined as follows:

$\underline{\mathsf{CLS+.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)}$

return $\mathsf{CGS.Sign}(gpk, bpk, \mathbf{gsk}[uid], m)$

$\underline{\mathsf{CLS+.Verify}(\texttt{tier-}1, gpk, bpk, m, \mu, \sigma)}$

return $\mathsf{CGS.Verify}(gpk, bpk, m, \mu, \sigma)$

Blind Conversions. The signatures can be blinded identically using the power of the CGS scheme. In CLS+.Convert, blinded signatures are now input and verified, leveraging the fact that even fully blinded inputs can be checked for their correctness. The signatures, pseudonyms, and blinded messages are re-randomised with CGS.RRandBlind to ensure non-transitivity. The pseudonyms are then blindly opened under the csk_1 using CGS.OpenBlind, and converted by raising them to the power of r and transforming them into the target group to ensure non-transitivity. The converted signature is simply a digital signature on the blinded converted pseudonym and message, with respect to the converter's verification key. In CLS+.Unblind the converted pseudonym and ciphertext are now unblinded with CGS.UnblindUser and CGS.UnblindM. The converted pseudonym, message and signature are included in the `tier`-2 signature, along with a proof that the unblinding has been done correctly. During `tier`-2 verification, the converter's signature on the blinded values and the proof of unblinding are verified.

$\mathsf{CLS+.Blind}(gpk, bpk, (m, \mu, \sigma))$

return $\mathsf{CGS.Blind}(gpk, bpk, (m, \mu, \sigma))$

$\mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k)$

$r \leftarrow\!\!\$\ \mathbb{Z}_p^*$, **for** $i = 1, \ldots k$:

$(c'_i, c\mu'_i, c\sigma'_i) \leftarrow\!\!\$\ \mathsf{CGS.RRandBlind}(gpk, bpk, c_i, c\mu_i, c\sigma_i)$

$c\mu''_i \leftarrow \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c'_i, c\mu'_i, c\sigma'_i)$

$c\mu'''_i \leftarrow e(g_1, c\mu''_i)^r, \sigma'_i \leftarrow\!\!\$\ \mathsf{SIG.Sign}(csk_2, (c'_i, c\mu'''_i, bpk))$

choose random permutation Π, **for** $i = 1, \ldots, k$: $(\overline{c\mu}_i, \overline{c}_i, \overline{c\sigma}_i) \leftarrow (c\mu'''_{\Pi(i)}, c'_{\Pi(i)}, \sigma'_{\Pi(i)})$

return $((\overline{c\mu}_1, \overline{c}_1, \overline{c\sigma}_1)), ..., (\overline{c\mu}_k, \overline{c}_k, \overline{c\sigma}_k)))$

$\mathsf{CLS+.Unblind}(bsk, (\overline{c\mu}, \overline{c\sigma}, \overline{c}))$

$\overline{\mu} \leftarrow \mathsf{CGS.UnblindUser}(bsk, \overline{c\mu}), m \leftarrow \mathsf{CGS.UnblindM}(bsk, \overline{c})$

$\pi_{\mathsf{ub}} \leftarrow\!\!\$\ \mathsf{SPK}\{bsk : \overline{\mu} = \mathsf{CGS.UnblindUser}(bsk, \overline{c\mu}) \wedge m = \mathsf{CGS.UnblindM}(bsk, \overline{c}) \wedge (bsk, bpk) \in \mathcal{BK}\}$

$\overline{\sigma} \leftarrow (\overline{c\mu}, \overline{c\sigma}, \overline{c}, \pi_{\mathsf{ub}})$ **return** $(\overline{\mu}, m, \overline{\sigma})$

CLS+.Verify(`tier`-2, $gpk, bpk, \overline{m}, \overline{\mu}, \overline{\sigma}$)

parse $\overline{\sigma} = (\overline{c\mu}, \overline{c\sigma}, \overline{c}, \pi_{\mathsf{ub}})$, Verify π_{ub} holds for $\overline{c\mu}, \overline{\mu}, \overline{c}, \overline{m}, bpk$

if SIG.Ver$((\overline{c\mu}, \overline{c}, bpk), cpk_2, \overline{c\sigma}) = 1$ **return** 1 **else return** 0

6.4 Security of CLS–CGS

We now show that our CLS–CGS construction satisfies all security properties defined in Section 6.2. More precisely, we show that the following theorem holds.

Theorem 6.1. The CLS–CGS construction presented in Section 6.3.1 is a secure CLS+ as defined in Section 6.2 if

- the CGS scheme is secure as defined in 5.3.2, and also satisfies extractability and the additional structural assumptions,
- the SIG is strongly EUF-cma secure,
- the SPK is zero–knowledge and sound,
- the DDH assumption holds in $\mathbb{G}_2$.

We now present proofs, showing that our CLS–CGS construction satisfies the correctness, anonymity, non–transitivity, conversion blindness, non–frameability and traceability requirements given in Section 6.2.

6.4.1 Correctness

Correctness of sign is clearly satisfied, due to the correctness of the commuting group signatures building block.

For all $i \in k$, the blinded signatures $c\mu_i, c\sigma_i, c_i$ are honestly generated and blinded commuting group signatures. In CLS+.Convert they are then re-randomised. Due to the re-randomisation property, they are indistinguishable from freshly generated and blinded

signatures. Let $cupk_i$ be the output of CGS.OpenBlind under the re-randomised blinded signatures, which will consist of elements in $\mathbb{G}_2$. The value $e(g_1, cupk_i)^r$ will be the pseudonym output by CLS+.Convert, and $\mathsf{CGS.UnblindUser}(bsk, e(g_1, cupk_i)^r)$ will be the pseudonym output by CLS+.Unblind. The commuting group signature satisfies:

$$\mathsf{CGS.UnblindUser}(bsk, e(g_1, cupk_i)^r) = e(g_1, \mathsf{CGS.UnblindUser}(bsk, cupk_i))^r.$$

Therefore due to the correctness of the commuting group signature scheme, letting upk_i be the user public key of the ith user, $\overline{\mu}_i = e(g_1, upk_i))^r$. Again, due to the correctness of the commuting group signatures, $\overline{m}_i = m_i$. Due to the correctness of the SPK and the digital signature used, the signature $\overline{\sigma}$ output will be valid. Therefore the construction satisfies **correctness of conversion**.

Assume

$$\mathsf{UnLink}(gpk, csk, ((\mu_0, m_0, \sigma_0), (\mu_1, m_1, \sigma_1))) = 0$$

$$\text{and } \mathsf{UnLink}(gpk, csk, ((\mu_1, m_1, \sigma_1), (\mu_2, m_2, \sigma_2))) = 0.$$

This ensures they are all valid blinded commuting group signatures, as otherwise UnLink would output $\perp$. For $i \in \{0, 1, 2\}$, let

$$upk_i = \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk, c_i, c\mu_i, c\sigma_i)).$$

Due to the same argument as correctness of conversion and the re-randomisation property, letting r_1, r_2 be the randomness chosen during Convert, $e(g_1, upk_0)^{r_1} = e(g_1, upk_1)^{r_1}$ and $e(g_1, upk_1)^{r_2} = e(g_1, upk_2)^{r_2}$. Therefore, $e(g_1, upk_0) = e(g_1, upk_1) = e(g_1, upk_2)$. However, if

$$\mathsf{UnLink}(gpk, csk, ((c\mu_0, c_0, c\sigma_0), (c\mu_2, c_2, c\sigma_2))) = 1,$$

then $e(g_1, upk_0)^{r_3} \neq e(g_1, upk_2)^{r_3}$, where r_3 was chosen during Convert. This is a contradiction. Therefore the construction satisfies **consistency**.

6.4.2 Anonymity

Lemma 6.1. The CLS–CGS construction satisfies **anonymity** if the DDH assumption holds in $\mathbb{G}_2$, and the CGS scheme satisfies anonymity, extractability, and the additional

$\mathsf{CONVERT}((c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk)$

if $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
if $\exists i^* \in [k], d \in \{0,1\}$ s.t $\mathsf{Identify}(uid_d^*, c_{i^*}, c\mu_{i^*}, c\sigma_{i^*}) = 1$
and $\mathsf{CGS.UnblindM}(bsk, c_{i^*}) = m^*$ and $bpk = bpk^*$
if $\exists i \in [k] \backslash \{i^*\}$ s.t. $\mathsf{Identify}(uid_d^*, c_i, c\mu_i, c\sigma_i) = 1$ for $d \in \{0,1\}$ **return** $\perp$
$(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{k-1}) \leftarrow_\$ \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i) : i \in [k] \backslash \{i^*\}\})$
if $(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{k-1}) = \perp$ **return** $\perp, \overline{c}_k \leftarrow_\$ \mathsf{CGS.RRandBlind}(gpk, bpk, c_{i^*})$
$upk^* \leftarrow_\$ \mathbb{G}_2, \overline{c\mu}_k \leftarrow e(g_1, \mathsf{CGS.BlindUser}(bpk, upk^*)), \overline{c\sigma}_k \leftarrow_\$ \mathsf{SIG.Sign}(csk_2, (\overline{c}_k, \overline{c\mu}_k, bpk))$
choose random permutation Π, **for** $i = 1, \ldots, k$:
$(\overline{c\mu}_i, \overline{c}_i, \overline{c\sigma}_i) \leftarrow (\overline{c\mu}_{\Pi(i)}, \overline{c}_{\Pi(i)}, \overline{c\sigma}_{\Pi(i)})$
else compute $\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k \leftarrow_\$ \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k)$
return $(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k)$

Figure 6.4: Convert oracle used during first j queries of Game $(0, j)$ in the CLS+ anonymity proof

structural assumptions.

Proof. We assume that an adversary $\mathcal{A}$ exists, that makes q queries to the SNDU oracle for distinct user identifiers, and q_{conv} queries to the CONVERT oracle, that wins in the anonymity game with probability $\epsilon + 1/2$.

We define Game (0,0) to be the anonymity experiment, with b chosen randomly at the beginning, using the CLS–CGS construction. Let $P_{0,0}$ be the event that an adversary $\mathcal{A}$ correctly guesses b after Game (0,0).

Game $(0, j)$ is identical to Game (0,0) except during the first j queries to the CONVERT oracle, when $c, c\mu, c\sigma$ is queried to the CONVERT oracle such that c unblinds to $\overline{m}$, for $d \in \{0,1\}$ $\mathsf{Identify}(uid_d^*, c, c\mu, c\sigma) = 1$ and $\overline{m} = m^*$. We provide the new convert oracle used for such j queries in Game $(0, j)$ in Figure 6.4. Let $P_{0,j}$ be the event that the adversary $\mathcal{A}$ correctly guesses b after Game $(0, j)$.

We show that Game (0,j) and Game (0, j+1) are indistinguishable assuming the DDH assumption in $\mathbb{G}_2$. We provide a distinguishing algorithm $\mathcal{D}_j$ in Figures 6.5 and 6.6, and then explain why $\mathcal{D}_j$ simulates inputs to $\mathcal{A}$ that are distributed identically to Game $(0, j)$ if a DDH tuple is input, and $\mathcal{D}_j$ simulates inputs to $\mathcal{A}$ that are distributed identically to Game $(0, j+1)$ if a DDH tuple is not input.

$\mathsf{CONVERT}((c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk)$

if $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
$s \leftarrow s + 1$ **if** $s \leq j$ perform CONVERT given in Figure 12
if $s > j + 1$ perform CONVERT given in anonymity experiment
If $\exists i \in [k]$ s.t $\mathsf{CGS.BlindVerify}(gpk, bpk, c_i, c\mu_i, c\sigma_i) = \perp$ **return** $\perp$
if $\exists i^* \in [k], d \in \{0,1\}$ s.t $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_{i^*}, c\mu_{i^*}, c\sigma_{i^*}))$
$= \mathbf{upk}[uid_d^*]$ and $\mathsf{CGS.UnblindM}(bsk, c_{i^*}) = m^*$ and $bpk = bpk^*$
if $uid_d^* \neq uid'$ abort $\mathcal{D}_j$
if $\exists i \in [k] \backslash \{i^*\}, d \in \{0,1\}$ s.t. $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_i, c\mu_i, c\sigma_i))$
$= \mathbf{upk}[uid_d^*]$ **return** $\perp$
for $i = [k] \backslash \{i^*\}$:
if $\exists uid \in \mathsf{HUL} \backslash \{uid'\}$ s.t $\mathsf{Identify}(uid, c_i, c\mu_i, c\sigma_i) = 1, upk_i' \leftarrow g_1^{\mathbf{usk}[uid]}$
else $upk_i' \leftarrow E_2(gpk, isk, osk, bsk, c_i, c\mu_i, c\sigma_i, \tau_e)$, **if** $upk_i' = \perp$ **return** $\perp$
$\overline{c}_i \leftarrow_\$ \mathsf{CGS.RRandBlind}(gpk, bpk, c_i), \overline{c\mu_i} \leftarrow e(upk_i', \mathsf{CGS.BlindUser}(bpk, D_3))$
$\overline{c\sigma}_i \leftarrow_\$ \mathsf{SIG.Sign}(csk_2, (\overline{c_i}, \overline{c\mu_i}, bpk))$
$\mu_{i^*} \leftarrow D_4, \overline{c\mu}_{i^*} \leftarrow e(g_1, \mathsf{CGS.BlindUser}(bpk, \mu_{i^*}))$
$\overline{c}_{i^*} \leftarrow_\$ \mathsf{CGS.RRandBlind}(gpk, bpk, c_{i^*}), \overline{c\sigma}_{i^*} \leftarrow_\$ \mathsf{SIG.Sign}(csk_2, (\overline{c}_{i^*}, \overline{c\mu}_{i^*}, bpk))$
choose random permutation Π, **for** $i = 1, \ldots, k$:
$(\overline{c\mu}_i, \overline{c}_i, \overline{c\sigma}_i) \leftarrow (\overline{c\mu}_{\Pi(i)}, \overline{c}_{\Pi(i)}, \overline{c\sigma}_{\Pi(i)})$
else compute $\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k \leftarrow_\$ \mathsf{CLS+.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k)$
return $(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k)$

Figure 6.5: $\mathcal{D}_j$ a distinguishing algorithm for the DDH problem in the CLS+ anonymity proof

$\mathsf{SNDU}(uid, n)$

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, l \leftarrow l+1, \mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
  if $l = k$ $uid' \leftarrow uid, \mathbf{upk}[uid] \leftarrow D_2, \mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk, \perp, \mathbf{upk}[uid])$ **return** $(\mathbf{upk}[uid], \mathbf{dec}^{uid})$
if $uid = uid'$
  if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow\!\!\$\ S_2(\mathsf{param}_{\mathsf{CGS}}, \mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}}, \tau_s)$, **return** $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$
Continue from line 5 of oracle in anonymity experiment

$\mathsf{SIGN}(uid, m, bpk)$

if $uid \neq uid'$ perform SIGN oracle from anonymity experiment
if $uid = uid'$
  if $\mathbf{dec}^{uid} \neq \mathsf{accept}$ **return** $\perp$
  $(\mu, \sigma) \leftarrow\!\!\$\ S_3(\mathsf{param}_{\mathsf{CGS}}, gpk, bpk, \mathbf{upk}[uid], m, \tau_s), \mathsf{SL} \leftarrow \mathsf{SL} \cup \{(uid, m, bpk)\}$
  return (σ, μ)

$\mathcal{D}_j(D_1, D_2, D_3, D_4)$

$s, l \leftarrow 0, k \leftarrow\!\!\$\ [1, q], b \leftarrow\!\!\$\ \{0,1\}, g_1 \leftarrow\!\!\$\ \mathbb{G}_1, g_2 \leftarrow D_1$
$(\mathsf{param}_{\mathsf{CGS}}, \tau_s, \tau_e) \leftarrow\!\!\$\ SE_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{sig}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}(1^\tau)$
$\mathsf{param} \leftarrow (\mathsf{param}_{\mathsf{CGS}}, \mathsf{param}_{\mathsf{sig}}), (isk, ipk) \leftarrow\!\!\$\ \mathsf{CLS{+}.IKGen}(\mathsf{param}), (csk, cpk) \leftarrow\!\!\$\ \mathsf{CLS{+}.CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$(uid_0^*, uid_1^*, m^*, bpk^*, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU,SIGN,CONVERT}}(\mathsf{choose}, gpk, isk)$
if $uid_0^* \notin \mathsf{HUL}$ or **if** $\mathbf{dec}^{uid_0^*} \neq \mathsf{accept}$ or $uid_1^* \notin \mathsf{HUL}$ or **if** $\mathbf{dec}^{uid_1^*} \neq \mathsf{accept}$ **if** $b = 0$ **return** 1 **else return** 0
$(\mu^*, \sigma^*) \leftarrow\!\!\$\ \mathsf{SIGN}(uid', m^*, bpk^*)$
$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU,SIGN,CONVERT}}(\mathsf{guess}, \mathsf{st}, \mu^*, \sigma^*)$
if $b^* = b$ **return** 1

Figure 6.6: $\mathcal{D}_j$ a distinguishing algorithm for the DDH problem in the CLS+ anonymity proof

The values gpk, csk, isk are distributed identically to the anonymity game, as everything but $g_2, \mathsf{param}_{\mathsf{CGS}}$ are chosen in the same way. The simulator SE_1 outputs a $\mathsf{param}_{\mathsf{CGS}}$ that is identically distributed to the output of CRSSetup.

Simulating the SNDU oracle. The SNDU oracle only differs from the oracle in the anonymity experiment when uid' is input. In this case, upk is distributed identically, and S_2 simulates the CGS.Join protocol. Therefore the outputs of SNDU are distributed identically to the anonymity experiment, due to the extractability of the commuting group signatures.

Simulating the SIGN oracle. The SIGN oracle is identical to the oracle in the anonymity experiment when $uid \neq uid'$ is queried. When uid' is queried, S_3 can then be used to simulate signatures that are identically distributed, due to the extractability of commuting group signatures. Therefore the outputs of SIGN are distributed identically to the anonymity experiment.

Simulating the CONVERT oracle. Other than the jth query, the CONVERT oracle is identical to both Games $(0, j)$ and $(0, j + 1)$.

For the jth query, if the input to $\mathcal{D}_j$ is a DDH tuple, then outputs from the CONVERT oracle are identically distributed to in the anonymity game. This is because if $c, c\mu, c\sigma$ is not queried to the CONVERT oracle such that c unblinds to $\overline{m}$, $\mathsf{Identify}(uid_d^*, c, c\mu, c\sigma) = 1$ for $d \in \{0, 1\}$ and $\overline{m} = m^*$ and $bpk = bpk^*$, the oracle behaves identically to the anonymity game. If an invalid blinded signature is input, the oracle will abort, as in the original anonymity game.

If this is queried, we show the simulation is correctly distributed. Firstly $\overline{c}_i$ and $\overline{c\sigma}_i$ are generated identically to CLS+.Convert.

As we do not have access to $\mathbf{gsk}[uid']$, we cannot perform Identify on input uid', therefore instead we perform CGS.OpenBlind and CGS.UnblindUser. Only one signature identifies to uid_d^*, otherwise the oracle will abort as in the anonymity game. We assume $uid_d^* = uid'$, which happens with probability $1/q$, therefore extraction of the upk_i' is always successful

because the signatures do not open to the user public key of uid'.

Letting $r' = \log_{D_1}(D_3)$ and $upk'_i = g_1^{usk_i}$, $upk_i = g_2^{usk_i}$, hence

$$\overline{c\mu}_i = e(upk'_i, \mathsf{CGS.BlindUser}(bpk, D_3)) = e(g_1, \mathsf{CGS.BlindUser}(bpk, D_3)^{usk_i})$$

$$= e(g_1, \mathsf{CGS.BlindUser}(bpk, g_2^{r'usk_i})) = e(g_1, \mathsf{CGS.BlindUser}(bpk, upk_i))^{r'},$$

which is correctly distributed. Also, letting $upk = \mathbf{upk}[uid']$,

$$\overline{c\mu}_{i^*} = e(g_1, \mathsf{CGS.BlindUser}(bpk, \mu_{i^*})) = e(g_1, \mathsf{CGS.BlindUser}(bpk, D_4))$$

$$= e(g_1, \mathsf{CGS.BlindUser}(bpk, upk^{r'})) = e(g_1, \mathsf{CGS.BlindUser}(bpk, upk))^{r'}.$$

As $\mathsf{CGS.BlindUser}$ on input upk is indistinguishable from the output of $\mathsf{CGS.RRandBlind}$ followed by $\mathsf{CGS.OpenBlind}$ on a blinded signature that would open to upk after unblinding, the converted pseudonyms are distributed correctly. The $\overline{c\mu}_i$ are then shuffled with a random permutation. Therefore, the outputs of the $\mathsf{CONVERT}$ oracle are distributed identically to the $\mathsf{CONVERT}$ oracle in the anonymity game, and so identically to Game $(0, j)$.

Simulating (μ^*, σ^*). The signature (μ^*, σ^*) input to $\mathcal{A}$ in the guessing stage is distributed identically to the anonymity game, due to outputs of the SIGN oracle being distributed identically to the anonymity game.

Reduction to the DDH problem. If the input to $\mathcal{D}_j$ is not a DDH tuple, then the output of the $\mathsf{CONVERT}$ oracle is identically distributed to Figure 6.4. This is because if $c, c\mu, c\sigma$ is not queried to the $\mathsf{CONVERT}$ oracle such that c unblinds to $\overline{m}$, $\mathsf{Identify}(uid^*_d, c, c\mu, c\sigma) = 1$ and $\overline{m} = m^*$ for $d \in \{0, 1\}$ and $bpk = bpk^*$, then the oracle behaves identically to both games. If this is the case, as D_4 is now chosen randomly and independently, $\overline{c}^*_i, \overline{c\mu}^*_i, \overline{c\sigma}^*_i$ are now chosen identically to Figure 6.4. Therefore, if the input to $\mathcal{D}_j$ is not a DDH tuple, then the outputs to $\mathcal{A}$ are identically distributed to Game $(0, j + 1)$.

The distinguisher $\mathcal{D}_j$ only aborts early if $c, c\mu, c\sigma$ is queried to the $\mathsf{CONVERT}$ oracle,

such that c unblinds to $\overline{m}$, $\mathsf{Identify}(uid_d^*, c, c\mu, c\sigma) = 1$ and $\overline{m} = m^*$, for $d \in \{0,1\}$ and $uid' \neq uid_d^*$. This occurs with probability at most $q - 1/q$. Therefore the probability that $\mathcal{D}_j$ outputs 1 given a DDH tuple was input is $\Pr[P_{0,j}]/q$. The probability that $\mathcal{D}_j$ outputs 1 given a DDH tuple was not input is $\Pr[P_{0,j+1}]/q$. The advantage of $\mathcal{D}_j$ is then $|\Pr[P_{0,j}] - \Pr[P_{0,j+1}]|/q$, therefore $|\Pr[P_{0,j}] - \Pr[P_{0,j+1}]| = q\epsilon_{\mathsf{DDH}}$.

We define Game 1 to be Game $(0, q_{\mathsf{conv}})$, where q_{conv} is the number of queries to the CONVERT oracle. Let P_1 be the event that an adversary $\mathcal{A}$ correctly guesses b after Game 1. As $|\Pr[P_{0,j}] - \Pr[P_{0,j+1}]| = q\epsilon_{\mathsf{DDH}}$, then $|\Pr[P_{0,0}] - \Pr[P_1]| \leq q_{\mathsf{conv}} q\epsilon_{\mathsf{DDH}}$.

Next, we show that $|\Pr[P_1] - 1/2| \leq \epsilon_{\mathsf{CGSanon}}$. We build an adversary $\mathcal{A}'$ that successfully guesses b in the anonymity game for commuting group signatures, given $\mathcal{A}$ that guesses successfully in Game 1 with $\Pr[P_1]$. We provide $\mathcal{A}'$ in Figure 6.7, and then explain why the simulation input to $\mathcal{A}$ is identically distributed to Game 1, and that $\mathcal{A}'$ successfully breaks the anonymity of commuting group signatures.

The keys (gpk, isk) are computed identically to the anonymity game. As the $\mathsf{SNDU}_{\mathsf{CGS}}$ oracle for the commuting group signatures join protocol is identical to the SNDU oracle for the CLS+ join protocol, the SNDU oracle is distributed identically. As users' secret keys and signatures are distributed identically, the SIGN oracle is distributed correctly, and input in the guessing phase is distributed correctly.

Simulating the CONVERT oracle. The CONVERT oracle can no longer use Identify without the secret keys of the honest users. However, if there exists $i^* \in [k]$ such that $\mathsf{Identify}(uid_d^*, c_{i^*}, c\mu_{i^*}, c\sigma_{i^*}) = 1$ for $d \in \{0,1\}$ and $\mathsf{CGS.UnblindM}(bsk, c_{i^*}) = m^*$ and $bpk = bsk^*$, then $\mathsf{BOPEN}(c_{i^*}, c\mu_{i^*}, c\sigma_{i^*}, bpk, bsk)$ would output $\perp$ in the CGS anonymity game. If there exists $i \in [k]$ such that $\mathsf{Identify}(uid_d^*, c_i, c\mu_i, c\sigma_i) = 1$ for $d \in \{0,1\}$, then BOPEN would either open to the user public key of uid_d^* or would output $\perp$. Therefore these conditions can be checked instead. Finally, the only difference to in Game 1 is that BOPEN is used in CLS+.Convert instead of CGS.OpenBlind; as BOPEN will not output $\perp$, this oracle is distributed identically to Figure 6.4.

$\underline{\mathsf{SNDU}(uid, M_{\mathsf{in}})}$

return $\mathsf{SNDU}_{\mathsf{CGS}}(uid, M_{\mathsf{in}})$

$\underline{\mathsf{SIGN}(uid, m, bpk)}$

return $\mathsf{SIGN}_{\mathsf{CGS}}(uid, m, bpk)$

$\underline{\mathsf{CONVERT}((c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk)}$

if $\exists i \in [1,k]$ s.t $\mathsf{CGS.BlindVerify}(gpk, bpk, c_i, c\mu_i, c\sigma_i) = 0$ or $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
if $\exists i^* \in [k]$ s.t $\mathsf{BOPEN}(c_{i^*}, c\mu_{i^*}, c\sigma_{i^*}, bpk, bsk) = \perp$
  if $\exists i \in [k]\backslash\{i^*\}$ s.t. $\mathsf{BOPEN}(c_i, c\mu_i, c\sigma_i, bpk, bsk) = \perp$ or $\mathbf{upk}[uid_d^*]$ for $d \in \{0,1\}$
    return $\perp$
  Using the BOPEN oracle instead of $\mathsf{CGS.OpenBlind}$ compute
  $(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{k-1}) \leftarrow\!\!\$\ \mathsf{CLS{+}.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i) : i \in [k]\backslash\{i^*\}\})$
  if $(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{k-1}) = \perp$ **return** $\perp$
  $\overline{c}_k \leftarrow\!\!\$\ \mathsf{CGS.RRandBlind}(gpk, bpk, c_{i^*}), upk^* \leftarrow\!\!\$\ \mathbb{G}_2, \overline{c\mu}_k \leftarrow e(g_1, \mathsf{CGS.BlindUser}(bpk, upk^*))$
  $\overline{c\sigma}_k \leftarrow\!\!\$\ \mathsf{SIG.Sign}(csk_2, (\overline{c}_k, \overline{c\mu}_k, bpk))$
  choose random permutation Π, **for** $i = 1, \ldots, k$:
  $(\overline{c\mu}_i, \overline{c}_i, \overline{c\sigma}_i) \leftarrow (\overline{c\mu}_{\Pi(i)}, \overline{c}_{\Pi(i)}, \overline{c\sigma}_{\Pi(i)})$
else Using the BOPEN oracle instead of $\mathsf{CGS.OpenBlind}$ compute
$\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k \leftarrow\!\!\$\ \mathsf{CLS{+}.Convert}(gpk, bpk, csk, \{(c\mu_i, c\sigma_i, c_i)\}_k)$
return $(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_k)$

$\underline{\mathcal{A}'^{\mathsf{SNDU}_{\mathsf{CGS}}, \mathsf{SIGN}_{\mathsf{CGS}}, \mathsf{OPEN}, \mathsf{BOPEN}}(\mathsf{choose}, gpk_{\mathsf{CGS}}, isk_{\mathsf{CGS}})}$

parse $gpk_{\mathsf{CGS}} = (\mathsf{param}_{\mathsf{CGS}}, opk_{\mathsf{CGS}}, ipk_{\mathsf{CGS}})$
$\mathsf{param}_{\mathsf{sig}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}(1^\tau), (cpk_2, csk_2) \leftarrow\!\!\$\ \mathsf{SIG.KeyGen}(1^\tau), \mathsf{param} \leftarrow (\mathsf{param}_{\mathsf{CGS}}, \mathsf{param}_{\mathsf{sig}})$
$(ipk, isk) \leftarrow (ipk_{\mathsf{CGS}}, isk_{\mathsf{CGS}}), cpk \leftarrow (opk_{\mathsf{CGS}}, cpk_2), gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$(uid_0^*, uid_1^*, m^*, bpk^*, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU}, \mathsf{SIGN}, \mathsf{CONVERT}}(\mathsf{choose}, gpk, isk)$
if $uid_0^* \notin \mathsf{HUL}$ or **if** $\mathbf{dec}^{uid_0^*} \neq \mathsf{accept}$ or $uid_1^* \notin \mathsf{HUL}$ or **if** $\mathbf{dec}^{uid_1^*} \neq \mathsf{accept}$ **return** 0
return $(uid_0^*, uid_1^*, m^*, bpk^*, \mathsf{st}_{\mathsf{CGS}})$

$\underline{\mathcal{A}'^{\mathsf{SNDU}, \mathsf{SIGN}, \mathsf{OPEN}, \mathsf{BOPEN}}(\mathsf{guess}, \mathsf{st}_{\mathsf{CGS}}, \mu^*, \sigma^*)}$

$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU}, \mathsf{SIGN}, \mathsf{CONVERT}}(\mathsf{guess}, \mathsf{st}, \mu^*, \sigma^*)$
return b^*

Figure 6.7: $\mathcal{A}'$ which breaks the anonymity of commuting group signatures, given $\mathcal{A}$ which breaks the anonymity of CLS–CGS

Reduction to the Anonymity of Commuting Group Signatures. As the inputs to $\mathcal{A}$ are distributed identically to Game 1, they will guess correctly with probability P_1 and so $\mathcal{A}'$ will guess correctly in the CGS anonymity game with probability $\Pr[P_1]$. Therefore, $|\Pr[P_1] - 1/2| \leq \epsilon_{\mathsf{CGSanon}}$. Therefore, $|\Pr[P_{0,0} - 1/2| \leq \epsilon_{\mathsf{CGSanon}} + q_{\mathsf{conv}} q \epsilon_{\mathsf{DDH}}$. As this is negligible, our construction satisfies anonymity.

□

6.4.3 Non–transitivity

Lemma 6.2. The CLS–CGS construction satisfies **non-transitivity** if the DDH assumption holds in $\mathbb{G}_2$, and the CGS scheme satisfies extractability and the additional structural assumptions.

Proof. For proving non-transitivity, we have to show that there exists an efficient simulator SIM that makes the real and simulated game indistinguishable. We start by describing the simulator and then explain why the real and simulated conversion oracles CONVERT and CONVSIM are indistinguishable.

$\mathsf{SIM}(gpk, bpk, csk, \mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k'}})$

$l \leftarrow 0, \forall j \in [1, k']$

$\quad \mu' \leftarrow_\$ \mathbb{G}_2; \forall c \in L_{uid_j}$

$\quad\quad l \leftarrow l + 1, \overline{c\mu}_l \leftarrow e(g_1, \mathsf{CGS.BlindUser}(bpk, \mu')), \bar{c}_l \leftarrow_\$ \mathsf{CGS.RRandBlind}(gpk, bpk, c)$

$\quad\quad \overline{c\sigma}_l \leftarrow_\$ \mathsf{SIG.Sign}(csk_2, (\overline{c\mu}_l, \bar{c}_l, bpk))$

return $((\overline{c\mu}_1, \bar{c}_1, \overline{c\sigma}_1), \ldots (\overline{c\mu}_l, \bar{c}_l, \overline{c\sigma}_l))$

We assume that an adversary $\mathcal{A}$ exists that makes q queries to the SNDU oracle for distinct user identifiers, and q_{conv} queries to the CONVX oracle, that guesses b correctly in the non-transitivity game with SIM as described and wins with probability $\epsilon + 1/2$.

We will stepwise make the real-world (b=0) and the simulated world (b=1) equivalent, using a sequence of Games $\mathcal{H}_j$ for $j = 0, \ldots, q$. The idea is that in Game $\mathcal{H}_j$ we will not

use simulated conversions for all users $uid_1, \ldots, uid_j$ in order of when they were queried to SNDU. More precisely, we define Game $\mathcal{H}_j$ to be as given in Figure 6.8 with all other oracles identical to the non–transitivity experiment. Let P_j be the event that $\mathcal{A}$ guesses b correctly in Game $\mathcal{H}_j$, with the simulator as described. Game $\mathcal{H}_j$ keeps track of the queries to SNDU, adding the first j queries of uid to a set UL. Then during queries to CONVSIM, if a signature of a user in UL is queried, these are treated in the same way as pseudonyms for corrupted users, i.e., they are normally converted using the CLS+.Convert algorithm. If a signature of an honest user that is not in UL is queried, we add this user to NUL. These conversions are simulated as usual.

Game $\mathcal{H}_0$ is identical to the non-transitivity game because UL is empty. Therefore, $\Pr[P_0] = \epsilon + 1/2$. In Game $\mathcal{H}_q$, UL contains all honest users, and so the CONVSIM oracle is now identical to the CONVERT oracle, and inputs to the adversary are now independent of b , therefore $\Pr[P_q] = 1/2$.

We now show that if an adversary can distinguish Games $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$, we can turn this into a distinguisher $\mathcal{D}_j$ that can break the DDH assumption in $\mathbb{G}_2$. We describe the reduction and the additional simulation that is needed in Figures 6.9 and 6.10. The CONVERT oracle remains unchanged. To avoid confusion, we write uid' to refer to the $j+1$-th user that has joined the group (and for which $\mathcal{D}_j$ embedded the DDH challenge).

We now argue that when a DDH tuple (D'_1, D'_2, D'_3, D'_4) is input to $\mathcal{D}_j$, the inputs to $\mathcal{A}$ are distributed identically to Game $\mathcal{H}_{j+1}$; when a DDH tuple is not input, the inputs to $\mathcal{A}$ are distributed identically to Game $\mathcal{H}_j$. That is for $D'_1 = h, D'_2 = h^a, D'_3 = h^b, D'_4 = h^c$, the oracles provided by $\mathcal{D}_j$ will be exactly as in $\mathcal{H}_{j+1}$ when $c = ab$, and as in $\mathcal{H}_j$ otherwise.

We first note that due to the DDH random self-reduction given in [127], if a DDH tuple is input to $\mathcal{D}_j$, then for all $i \in [q_{\mathsf{conv}}]$, $D_1, D_2, D_{3,i}, D_{4,i}$ is a DDH tuple. If a DDH tuple is not input to $\mathcal{D}_j$, then $D_1, D_2, D_{3,1}, D_{4,1}, \cdots, D_{3,q_{\mathsf{conv}}}, D_{4,q_{\mathsf{conv}}}$ is randomly and independently distributed.

The values gpk, csk, isk are distributed identically to the non-transitivity game, as everything but $g_2, \mathsf{param}_{\mathsf{CGS}}$ are chosen in the same way. SE_1 and Setup outputs identically distributed $\mathsf{param}_{\mathsf{CGS}}$.

Game $\mathcal{H}_j$

$t \leftarrow 0, b \leftarrow_\$ \{0,1\}, \mathsf{param} \leftarrow_\$ \mathsf{CLS+.Setup}(1^\tau), (ipk, isk) \leftarrow_\$ \mathsf{CLS+.IKGen}(\mathsf{param})$
$(cpk, csk) \leftarrow_\$ \mathsf{CLS+.CKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, cpk)$
return $\mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVX}}(\mathsf{guess}, gpk, isk)$

$\mathsf{SNDU}(uid, M_{\mathsf{in}})$

if $uid \notin \mathsf{HUL}, t \leftarrow t+1,$ **if** $t \leq j \quad \mathsf{UL} \leftarrow \mathsf{UL} \cup \{uid\}$
Continue from line 2 of standard SNDU oracle

$\mathsf{CONVSIM}((c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk)$

if $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
if $\mathsf{CLS+.Convert}(gpk, bpk, csk, (c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k)) = \perp$ **return** $\perp$
Set $\mathsf{CL} \leftarrow \emptyset$
$\forall i \in [1,k]$
  if $\exists uid \in \mathsf{HUL}$ s.t $\mathsf{Identify}(uid, c_i, c\mu_i, c\sigma_i) = 1$
    if L_{uid} does not exist, create $\mathsf{L}_{uid} \leftarrow \{c_i\}, \mathsf{CL}_{uid} \leftarrow \{(c_i, c\sigma_i, c\mu_i)\}$
    else set $\mathsf{L}_{uid} \leftarrow \mathsf{L}_{uid} \cup \{c_i\}, \mathsf{CL}_{uid} \leftarrow \mathsf{CL}_{uid} \cup \{(c_i, c\sigma_i, c\mu_i)\}$
  else $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(c_i, c\sigma_i, c\mu_i)\}$
$(\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{i=1,\ldots k'}) \leftarrow_\$ \mathsf{CLS+.Convert}(gpk, bpk, csk, \mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid}))$
for $k' \leftarrow |\mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid}|$
Let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters
Let $\mathsf{NUL} \leftarrow \{uid_1, \ldots uid_{k''}\} \backslash \mathsf{UL}$
$\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{i=k'+1,\ldots k} \leftarrow_\$ \mathsf{SIM}(gpk, bpk, csk, \bigcup_{uid \in \mathsf{NUL}} \mathsf{L}_{uid})$
Let $\{(\overline{c\mu}'_i, \overline{c\sigma}'_i, \overline{c}'_i)\}_{i=1,\ldots k}$ be a random permutation of $\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{i=1,\ldots k}$
return $(\{(\overline{c\mu}'_i, \overline{c\sigma}'_i, \overline{c}'_i)\}_{i=1,\ldots k})$

Figure 6.8: Description of Game $\mathcal{H}_j$ and the changes to the SNDU and CONVSIM oracles in the CLS+ non–transitivity proof

$\mathcal{D}_j(D'_1, D'_2, D'_3, D'_4)$

Expand D'_1, D'_2, D'_3, D'_4 into $D_1, D_2, \{D_{3,i}, D_{4,i} : i \in [q_{\mathsf{conv}}]\}$ using DDH random self-reduction [127]
$s, t \leftarrow 0, g_1 \leftarrow_\$ \mathbb{G}_1, g_2 \leftarrow D_1, (\mathsf{param}_{\mathsf{CGS}}, \tau_s, \tau_e) \leftarrow_\$ SE_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$
$\mathsf{param}_{\mathsf{sig}} \leftarrow_\$ \mathsf{SIG.Setup}(1^\tau), \mathsf{param} \leftarrow (\mathsf{param}_{\mathsf{CGS}}, \mathsf{param}_{\mathsf{sig}}), (isk, ipk) \leftarrow_\$ \mathsf{CLS+.IKGen}(\mathsf{param})$
$(csk, cpk) \leftarrow_\$ \mathsf{CLS+.CKGen}(\mathsf{param}), gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVX}}(gpk, csk, isk)$
if $b^* = b$ **return** 1 **else return** 0

$\mathsf{SNDU}(uid, n)$

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}, t \leftarrow t + 1,$ **if** $t \leq j$ $\mathsf{UL} \leftarrow \mathsf{UL} \cup \{uid\}$
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
  if $t = j + 1$ $uid' \leftarrow uid, \mathbf{upk}[uid] \leftarrow D_2, \mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow (gpk, \perp, \mathbf{upk}[uid])$ **return** $(D_2, \mathbf{dec}^{uid})$
if $uid = uid'$
  if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow_\$ S_2(\mathsf{param}_{\mathsf{CGS}}, \mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}}, \tau_s),$ **return** $(M_{\mathsf{out}}, \mathbf{dec}^{uid})$
Continue from line 5 of standard SNDU oracle

$\mathsf{SIGN}(uid, m, bpk)$

if $uid \neq uid'$ perform SIGN oracle from non–transitivity experiment
if $uid = uid'$
  if $\mathbf{dec}^{uid} \neq \mathsf{accept}$ **return** $\perp$
  $(\mu, \sigma) \leftarrow_\$ S_3(\mathsf{param}_{\mathsf{CGS}}, gpk, bpk, \mathbf{upk}[uid], m, \tau_s), \mathsf{SL} \leftarrow \mathsf{SL} \cup \{(uid, m, bpk)\}$
  return (σ, μ)

Figure 6.9: $\mathcal{D}_j$ our distinguishing algorithm for the DDH problem in the CLS+ non–transitivity proof

Simulating the SNDU oracle. The SNDU oracle only differs from the oracle in the non–transitivity experiment during the $(j + 1)$-th query by embedding D_2 of the DDH challenger into the user's public key upk. Clearly, upk is distributed correctly, and CGS.Join can be simulated with S_2 on input upk using τ_s. Note that usk, upk' are not defined for this user, but this is not output to $\mathcal{A}$, or used in the next stage of the protocol.

Simulating the SIGN oracle. The SIGN oracle is identical to the oracle in the non–transitivity experiment when $uid \neq uid'$ is queried. When uid' is queried, we simply use S_3 on input $\mathbf{upk}[uid]$ to simulate signatures.

Simulating the CONVSIM oracle. What remains to be shown is that the CONVSIM oracle either behaves identically to the CONVSIM oracle in Game $\mathcal{H}_j$ or as in $\mathcal{H}_{j+1}$,

$\mathsf{CONVSIM}((c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk)$

$s \leftarrow s+1$, **if** $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
if $\mathsf{CLS+.Convert}(gpk, bpk, csk, (c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k)) = \perp$ **return** $\perp$
Set $\mathsf{CL} \leftarrow \emptyset$
$\forall i \in [1,k]$
  if $\exists uid \in \mathsf{HUL}$ s.t $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) = \mathbf{upk}[uid]$
    if L_{uid} does not exist $\mathsf{L}_{uid} \leftarrow \{c_i\}, \mathsf{CL}_{uid} \leftarrow \{(c_i, g_1^{\mathbf{usk}[uid]})\}$
    else $L_{uid} \leftarrow \mathsf{L}_{uid} \cup \{c_i\}, \mathsf{CL}_{uid} \leftarrow \mathsf{CL}_{uid} \cup \{(c_i, g_1^{\mathbf{usk}[uid]})\}$
  else $upk'_i \leftarrow E_2(gpk, bpk, isk, osk, bsk, c_i, c\mu_i, c\sigma_i, \tau_e)$
    $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(c_i, upk'_i)\}$
$n \leftarrow 0$; $\forall (c, upk') \in \mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid}$
  $n \leftarrow n+1, \overline{c\mu}_n \leftarrow e(upk', \mathsf{CGS.BlindUser}(bpk, D_{3,s})), \overline{c}_n \leftarrow_\$ \mathsf{CGS.RRandBlind}(gpk, bpk, c)$
  $\overline{c\sigma}_n \leftarrow_\$ \mathsf{Sig}(csk, (\overline{c\mu}_n, \overline{c}_n, bpk))$
if $\mathsf{L}_{uid'}$ exists $\forall c \in L_{uid'}$
  $n \leftarrow n+1, \overline{c\mu}_n \leftarrow e(g_1, \mathsf{CGS.BlindUser}(bpk, D_{4,s})), \overline{c}_n \leftarrow_\$ \mathsf{CGS.RRandBlind}(gpk, bpk, c)$
  $\overline{c\sigma}_n \leftarrow_\$ \mathsf{Sig}(csk, (\overline{c\mu}_n, \overline{c}_n, bpk))$
let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters
Let $\mathsf{NUL} \leftarrow \{uid_1, \ldots uid_{k''}\} \backslash \mathsf{UL}$
$\{(\overline{c\mu}_i, \overline{c}_i, \overline{c\sigma}_i)\}_{i=n+1,\ldots k} \leftarrow_\$ \mathsf{SIM}(gpk, bpk, csk \bigcup_{uid \in \mathsf{NUL}, uid \neq uid'} \mathsf{L}_{uid})$
Let $\{(\overline{c\mu}'_i, \overline{c\sigma}'_i, \overline{c}'_i)\}_{i=1,\ldots k}$ be a random permutation of $\{(\overline{c\mu}_i, \overline{c\sigma}_i, \overline{c}_i)\}_{i=1,\ldots k}$
return $(\{(\overline{c\mu}'_i, \overline{c\sigma}'_i, \overline{c}'_i)\}_{i=1,\ldots k})$

Figure 6.10: The CONVSIM oracle used by distinguisher $\mathcal{D}_j$ in the CLS+ non–transitivity proof

depending on whether its input was a DDH tuple or not. The value upk' is only extracted if $c_i, c\mu_i, c\sigma_i$ does not identify to any honest user, and therefore this will be successful as it will not identify to $\mathbf{upk}[uid']$. In the case of uid' $\mathbf{usk}[uid']$ is not defined, however because $uid' \notin \mathsf{UL}$ then $\mathsf{CL}_{uid'}$ will never be used. The values $\overline{c}, \overline{c\sigma}$ are computed in the same way as in CLS+.Convert. We know that $D_{3,s} = g_2^{\tilde{r}}$ for some $\tilde{r}$, and $upk' = g_1^{usk}$ for some usk, and thus it must hold that:

$$\overline{c\mu} = e(upk', \mathsf{CGS.BlindUser}(bpk, D_{3,s})) = e(g_1, \mathsf{CGS.BlindUser}(bpk, g_2^{\tilde{r}}))^{usk}$$

$$= e(g_1, \mathsf{CGS.BlindUser}(bpk, upk))^{\tilde{r}}.$$

As CGS.BlindUser on input upk is indistinguishable from the output of CGS.RRandBlind followed by CGS.OpenBlind on a blinded signature that would open to upk after unblinding, these pseudonyms are distributed identically to the non–transitivity experiment.

If (D'_1, D'_2, D'_3, D'_4) is a DDH tuple, then $D_{4,s} = D_2^{\tilde{r}}$. Therefore as $upk_{uid'} = D_2$, and as $\mathsf{CGS.BlindUser}(bpk, upk_{uid'}^{\tilde{r}}) = \mathsf{CGS.BlindUser}(bpk, upk_{uid'})^{\tilde{r}}$, the inputs to $\mathcal{A}$ are also distributed identically to Game $\mathcal{H}_{j+1}$. Whereas if (D'_1, D'_2, D'_3, D'_4) is *not* a DDH tuple,

then $D_{4,s}$, is distributed identically to μ', which was chosen randomly and independently. Therefore the inputs to $\mathcal{A}$ are distributed identically to Game $\mathcal{H}_j$.

Reduction to the DDH problem. Therefore the probability that $\mathcal{D}_j$ outputs 1 if it was given a valid DDH tuple as input is $\Pr[P_{j+1}]$, and $\Pr[P_j]$ is the probability that $\mathcal{D}_j$ outputs 1 when the input was not a DDH tuple. The advantage of $\mathcal{D}_j$ is then $|\Pr[P_j] - \Pr[P_{j+1}]|$, therefore $|\Pr[P_j] - \Pr[P_{j+1}]| = \epsilon_{\mathsf{DDH}}$.

Overall, for our sequence of games $\mathcal{H}_0$ to $\mathcal{H}_q$ it holds that $|\Pr[P_0] - \Pr[P_q]| \leq q\epsilon_{\mathsf{DDH}}$ and thus $\epsilon \leq q\epsilon_{\mathsf{DDH}}$ is negligible. This concludes our proof that the CLS–CGS construction satisfies non–transitivity.

□

6.4.4 Conversion Blindness

Lemma 6.3. The CLS–CGS construction satisfies **conversion blindness** if the CGS scheme satisfies the blindness requirement and the additional structural assumptions and the SPK is zero–knowledge.

Proof. We build an adversary $\mathcal{A}'$ that successfully guesses b in the blindness game for commuting group signatures, given $\mathcal{A}$ that successfully guesses b in the conversion blindness game. We provide $\mathcal{A}'$ in Figure 6.11, and then explain why the simulation input to $\mathcal{A}$ is identically distributed to the conversion blindness experiment, and that $\mathcal{A}'$ successfully breaks the blindness of commuting group signatures.

The values (gpk, bpk, isk, csk) are computed identically to the conversion blindness game. As signatures and the blinding algorithm are identical for both the CLS–CGS construction and commuting group signatures, $c\mu^*, c\sigma^*, c^*$ is identically distributed to the conversion blindness game, with the same b chosen as that $\mathcal{A}'$ must guess.

Simulating the UNBLIND oracle. Firstly $\overline{\mu}_i = e(g_1, \mathsf{CGS.Open}(gpk, bpk, osk_{\mathsf{CGS}}, m_i, \mu_i, \sigma_i))^r$. This is distributed correctly because:

$\mathsf{UNBLIND}(\mu_1, \sigma_1, m_1), \ldots, (\mu_k, \sigma_k, m_k))$

if $\exists i \in [k]$ s.t $\mathsf{CLS+.Verify}(\texttt{tier-1}, gpk, bpk, m_i, \mu_i, \sigma_i) = 0$ **return** $\perp$
$r \leftarrow\!\!\$\ \mathbb{Z}_p^*, \forall i \in [1, k]$
  $(c\mu_i, c\sigma_i, c_i) \leftarrow\!\!\$\ \mathsf{CGS.Blind}(gpk, bpk, m_i, \mu_i, \sigma_i; r_i')$
  $(\overline{c_i}, \cdot, \cdot) \leftarrow\!\!\$\ \mathsf{CGS.RRandBlind}(gpk, bpk, c_i, c\mu_i, c\sigma_i; r_i'')$
  $\overline{\mu}_i \leftarrow e(g_1, \mathsf{CGS.Open}(gpk, bpk, osk_{\mathsf{CGS}}, m_i, \mu_i, \sigma_i))^r$
  $\overline{c\mu}_i \leftarrow e(g_1, \mathsf{CGS.BlindUser}(bpk, \mathsf{CGS.Open}(gpk, bpk, osk_{\mathsf{CGS}}, m_i, \mu_i, \sigma_i); r_i' + r_i''))^r$
  $\overline{c\sigma} \leftarrow\!\!\$\ \mathsf{SIG.Sign}(csk_2, (\overline{c_i}, \overline{c\mu}_i, bpk))$
  Simulate π_i with $\overline{\mu}_i, \overline{c\mu}_i, m_i, \overline{c_i}$
  $\overline{\sigma_i} \leftarrow (\overline{c\mu_i}, \overline{c\sigma}_i, \overline{c_i}, \pi_i)$
choose random permutation Π, **for** $i = 1, \ldots, k : (\overline{\mu}_i, \overline{\sigma_i}, m_i) \leftarrow (\overline{\mu}_{\Pi(i)}, \overline{\sigma_{\Pi(i)}}, m_{\Pi(i)})$
return $(\{(\overline{\mu}_i, \overline{\sigma_i}, m_i)\}_k, \{r_i'\}_k, \{r_i''\}_k, r, \Pi)$

$\mathcal{A}'(\mathsf{choose}, gpk_{\mathsf{CGS}}, bpk_{\mathsf{CGS}}, isk_{\mathsf{CGS}}, osk_{\mathsf{CGS}})$

parse $gpk_{\mathsf{CGS}} = (\mathsf{param}_{\mathsf{CGS}}, opk_{\mathsf{CGS}}, ipk_{\mathsf{CGS}})$
$\mathsf{param}_{\mathsf{sig}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}(1^\tau), (cpk_2, csk_2) \leftarrow\!\!\$\ \mathsf{SIG.KeyGen}(1^\tau)$
$\mathsf{param} \leftarrow (\mathsf{param}_{\mathsf{CGS}}, \mathsf{param}_{\mathsf{sig}}), (ipk, isk) \leftarrow (ipk_{\mathsf{CGS}}, isk_{\mathsf{CGS}}), bpk \leftarrow bpk_{\mathsf{CGS}}$
$(cpk, csk) \leftarrow ((opk_{\mathsf{CGS}}, cpk_2), (osk_{\mathsf{CGS}}, csk_2)), gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$(\mathsf{st}, (\mu_0, \sigma_0, m_0), (\mu_1, \sigma_1, m_1)) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{UNBLIND}}(\mathsf{choose}, gpk, bpk, isk, csk)$
return $(\mathsf{st}_{\mathsf{CGS}}, (\mu_0, \sigma_0, m_0), (\mu_1, \sigma_1, m_1))$

$\mathcal{A}'(\mathsf{guess}, \mathsf{st}_{\mathsf{CGS}}, c^*, c\mu^*, c\sigma^*)$

$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{UNBLIND}}(\mathsf{guess}, \mathsf{st}, c\mu^*, c\sigma^*, c^*)$, **return** b^*

Figure 6.11: $\mathcal{A}'$ which breaks the blindness of commuting group signatures using $\mathcal{A}$ which breaks the conversion blindness of CLS–CGS

$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk_{\mathsf{CGS}}, \mathsf{CGS.Blind}(gpk, bpk, , m_i, \mu_i, \sigma_i))) = \mathsf{CGS.Open}(gpk, bpk, osk_{\mathsf{CGS}}, m_i, \mu_i, \sigma_i))$;

and because $e(g_1, \mathsf{CGS.UnblindUser}(bsk, cupk))^r = \mathsf{CGS.UnblindUser}(bsk, e(g_1, cupk)^r)$.

The value $\overline{c_i}$ is computed identically to the original oracle. $\overline{c\mu}_i$ is distributed correctly because $\mathsf{CGS.BlindUser}(bpk, (\mathsf{CGS.Open}(gpk, bpk, osk, m, \mu, \sigma))) \approx \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c, c\mu, c\sigma)$. The randomness input to CGS.BlindUser corresponds to the randomness used to blind the pseudonym in Blind, and then to re-randomise the pseudonym in CGS.RRandBlind. The signature $\overline{c\sigma}$ is computed identically to CLS+.Convert. Finally, π_i can be simulated due to the zero–knowledge property of the SPK. Therefore, all outputs of UNBLIND are distributed identically to the experiment.

Reduction to the Blindness of Commuting Group Signature. If $\mathcal{A}$ correctly guesses b then $\mathcal{A}'$ correctly guesses b, and therefore blindness of commuting group signatures implies conversion blindness.

□

6.4.5 Non–frameability

Lemma 6.4. The CLS–CGS construction satisfies **non-frameability** if the CGS scheme satisfies the non–frameability requirement and the additional structural assumptions, SIG is strongly EUF-cma secure and the SPK is sound.

Proof. We build an adversary $\mathcal{A}'$ that successfully wins the non-frameability game for commuting group signatures, given $\mathcal{A}$ that wins the non–frameability game for the CLS–CGS construction. We provide $\mathcal{A}'$ in Figure 6.12 and then explain why the simulation input to $\mathcal{A}$ given in Figure 6.12 is identically distributed to the non–frameability experiment for the CLS–CGS construction, and that $\mathcal{A}'$ successfully breaks the non–frameability of commuting group signatures.

The values (gpk, isk) are computed identically to the non–frameability game. As the $\mathsf{SNDU}_{\mathsf{CGS}}$ oracle for the commuting group signatures join protocol is identical to the SNDU

$\underline{\mathsf{SNDU}(uid, M_{\mathsf{in}})}$

return $\mathsf{SNDU}_{\mathsf{CGS}}(uid, M_{\mathsf{in}})$

$\underline{\mathsf{SIGN}(uid, m, bpk)}$

return $\mathsf{SIGN}_{\mathsf{CGS}}(uid, m, bpk)$

$\underline{\mathsf{CONVERT}(c\mu_1, c\sigma_1, c_1), \ldots, (c\mu_k, c\sigma_k, c_k), bpk, bsk)}$

Identical to non–frameability experiment

$\mathcal{A}'^{\mathsf{SNDU}_{\mathsf{CGS}}, \mathsf{SIGN}_{\mathsf{CGS}}}(gpk_{\mathsf{CGS}}, isk_{\mathsf{CGS}}, osk_{\mathsf{CGS}})$

parse $gpk_{\mathsf{CGS}} = (\mathsf{param}_{\mathsf{CGS}}, opk_{\mathsf{CGS}}, ipk_{\mathsf{CGS}})$
$\mathsf{param}_{\mathsf{sig}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}(1^\tau), (cpk_2, csk_2) \leftarrow\!\!\$\ \mathsf{SIG.KeyGen}(1^\tau), \mathsf{param} \leftarrow (\mathsf{param}_{\mathsf{CGS}}, \mathsf{param}_{\mathsf{sig}})$
$(ipk, isk) \leftarrow (ipk_{\mathsf{CGS}}, isk_{\mathsf{CGS}}), (cpk, csk) \leftarrow ((opk_{\mathsf{CGS}}, cpk_2), (osk_{\mathsf{CGS}}, csk_2)), gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$(\overline{m}_0, \overline{\mu}_0, \overline{\sigma}_0, \overline{m}_1, \overline{\mu}_1, \overline{\sigma}_1, bpk, bsk) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SNDU}, \mathsf{SIGN}, \mathsf{CONVERT}}(gpk, isk)$
if $(bpk, bsk) \notin \mathcal{BK}$ **return** 0
$\forall b \in \{0,1\}$ **if** $\nexists(\overline{\mu}_b, \overline{m}_b, c_b, c\mu_b, c\sigma_b) \in \mathsf{UBL}$ **return** $\perp$
if $\nexists uid^* \in \mathsf{HUL}$ s.t $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk_{\mathsf{CGS}}, c_0, c\mu_0, c\sigma_0)) = \mathbf{upk}[uid^*]$
return $\perp$
if $\exists b \in \{0,1\}$ s,t $(uid^*, \overline{m}_b, bpk) \notin \mathsf{SL}$ **return** $(uid^*, c_b, c\mu_b, c\sigma_b, bpk, bsk)$ **else return** $\perp$

Figure 6.12: $\mathcal{A}'$ which breaks the non–frameability of commuting group signatures using $\mathcal{A}$ which breaks the non–frameability of CLS–CGS

oracle for the CLS+ construction join protocol, the SNDU oracle is distributed identically. As users' secret keys and signatures are distributed identically to commuting group signatures, the SIGN oracle is distributed correctly. The CONVERT oracle is identical to the non–frameability game.

Reduction to the Non–frameability of Commuting Group Signatures. We assume $\mathcal{A}$ is successful. Firstly, we consider the case that $\mathcal{A}'$ aborts due to $(\overline{\mu}_b, \overline{m}_b, c_b, c\mu_b, c\sigma_b)$ not being found in UBL. In this case, clearly $\mathcal{A}'$ could break the strong EUF-cma security of the digital signatures or the soundness of the SPK. Let $\overline{\sigma}_b = (\overline{c\mu}_b, \overline{c\sigma}_b, \overline{c}_b, \pi_{\mathsf{ub},b})$. The values $\overline{c\mu}_b, \overline{c\sigma}_b, \overline{c}_b$ were output by a convert query under bpk otherwise $\mathcal{A}'$ could break the strong EUF-cma security of the digital signatures. Due to the soundness of the SPK, $\mathsf{CGS.UnblindUser}(bsk, \overline{c\mu}_b) = \overline{\mu}_b$ and $\mathsf{CGS.UnblindM}(bsk, \overline{c}_b) = \overline{m}_b$. Therefore, $(\overline{\mu}_b, \overline{m}_b, c_b, c\mu_b, c\sigma_b)$ is not stored in UBL with at most negligible probability.

$\mathcal{A}'$ does not abort in the next step. This is because $\mathcal{A}$ is successful, and so $\exists uid^* \in \mathsf{HUL}$ such that $\mathsf{Identify}(bpk, bsk, uid^*, c_0, c\mu_0, c\sigma_0) = 1$. Again as $\mathcal{A}$ is successful, $\mathcal{A}'$ does not abort in the final step.

We now argue that $\mathcal{A}'$ successfully breaks the non–frameability of commuting group signatures. In order to do so, $c_b, c\mu_b, c\sigma_b$ output must be valid, which is clearly true otherwise the CONVERT oracle would have failed, and the signatures would not be stored in UBL. For $b \in \{0,1\}$ such that $(uid^*, \overline{m}_b, bpk) \notin \mathsf{SL}$, due to the re-randomisation property

$$(uid^*, \mathsf{CGS.UnblindM}(bsk, c_b) = (uid^*, \mathsf{CGS.UnblindM}(bsk, \overline{c}_b)),$$

thus $(uid^*, \mathsf{CGS.UnblindM}(bsk, c_b), bpk) \notin \mathsf{SL}$.

Finally we need to show that

$$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_b, c\mu_b, c\sigma_b)) = \mathbf{upk}[uid^*].$$

We already have that:

$$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_0, c\mu_0, c\sigma_0)) = \mathbf{upk}[uid^*].$$

Also,

$$\overline{\mu}_0 = \mathsf{CGS.UnblindUser}(bsk, e(g_1, \mathsf{CGS.OpenBlind}(gpk, bpk, c_0, c\mu_0, c\sigma_0))^{r_1}),$$

$$\overline{\mu}_1 = \mathsf{CGS.UnblindUser}(bsk, e(g_1, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_1, c\mu_1, c\sigma_1))^{r_2}),$$

where r_1, r_2 are the conversion randomness chosen in the CONVERT oracle. The probability that two converted pseudonyms will be output that unblind to the same pseudonym in different convert queries is negligible, as the adversary has no control over the conversion randomness. Therefore,

$$\mathsf{CGS.UnblindUser}(bsk, e(g_1, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_0, c\mu_0, c\sigma_0))^{r_1})$$

$$= \mathsf{CGS.UnblindUser}(bsk, e(g_1, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_1, c\mu_1, c\sigma_1))^{r_1}).$$

Therefore due to the requirements at the beginning of Section 6.3.1:

$$e(g_1, \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_0, c\mu_0, c\sigma_0))^{r_1}$$

$$= e(g_1, \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_1, c\mu_1, c\sigma_1))^{r_1}.$$

Therefore
$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, osk, c_1, c\mu_1, c\sigma_1)) = \mathbf{upk}[uid^*]$.

Therefore $\mathcal{A}'$ successfully breaks the non–frameability of commuting group signatures with probability $(1 - \mathsf{negl})\epsilon$. Therefore, assuming the non–frameability of commuting group signatures, the CLS–CGS construction satisfies non–frameability.

□

6.4.6 Traceability

Lemma 6.5. The CLS–CGS construction satisfies (`tier`-1 and `tier`-2) **traceability** if the CGS scheme satisfies the non–frameability and traceability requirements and the additional structural assumptions.

Proof. We build an adversary $\mathcal{A}'$ that successfully wins the traceability game for commuting group signatures, given $\mathcal{A}$ that wins the `tier`-2 traceability game for the CLS–CGS construction. We provide $\mathcal{A}'$ in Figure 6.13, and then explain why the simulation input to $\mathcal{A}$ is identically distributed to the `tier`-2 traceability experiment for the CLS–CGS construction, and that $\mathcal{A}'$ successfully breaks the traceability of commuting group signatures.

The values (gpk, csk) are distributed identically to the `tier`-2 traceability game. The $\mathsf{SNDI}_{\mathsf{CGS}}$ oracle for the commuting group signatures join protocol is identical to the SNDI oracle for the CLS+ join protocol, except that in the latter the user sends their user public key as their first message, whereas in the former the user's public key is part of the issuer's input. Therefore, the first message input to the SNDI oracle per user will be a user public key. This can be input to $\mathsf{SNDI}_{\mathsf{CGS}}$ with an empty message. Therefore, the SNDI oracle is distributed correctly. As users' secret keys and signatures are distributed identically, the SIGN oracle is distributed correctly. The $\mathsf{ADDU}_{\mathsf{CGS}}$ and ADDU oracles are identical.

Reduction to the Traceability of Commuting Group Signatures. As $\mathcal{A}$ is successful, all the blinded outputs are valid, therefore clearly provided $\mathcal{A}'$ can find $i \in$

$\underline{\mathsf{ADDU}(uid)}$

return $\mathsf{ADDU}_{\mathsf{CGS}}(uid)$

$\underline{\mathsf{SIGN}(uid, m, bpk)}$

$\mathsf{SIGN}_{\mathsf{CGS}}(uid, m, bpk)$

$\underline{\mathsf{SNDI}(uid, M_{\mathsf{in}})}$

Parse $M_{\mathsf{in}} = upk$
return $\mathsf{SNDI}_{\mathsf{CGS}}(uid, \perp, upk)$

$\underline{\mathcal{A}'^{\mathsf{ADDU}_{\mathsf{CGS}},\mathsf{SNDI}_{\mathsf{CGS}},\mathsf{SIGN}_{\mathsf{CGS}}}(gpk_{\mathsf{CGS}}, osk_{\mathsf{CGS}})}$

Parse $gpk_{\mathsf{CGS}} = (\mathsf{param}_{\mathsf{CGS}}, opk_{\mathsf{CGS}}, ipk_{\mathsf{CGS}}), \mathsf{param}_{\mathsf{sig}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}(1^\tau)$
$(cpk_2, csk_2) \leftarrow\!\!\$\ \mathsf{SIG.KeyGen}(1^\tau), \mathsf{param} \leftarrow (\mathsf{param}_{\mathsf{CGS}}, \mathsf{param}_{\mathsf{sig}}), ipk \leftarrow ipk_{\mathsf{CGS}}$
$(cpk, csk) \leftarrow ((opk_{\mathsf{CGS}}, cpk_2), (osk_{\mathsf{CGS}}, csk_2)), gpk \leftarrow (\mathsf{param}, ipk, cpk)$
$((c_1, c\mu_1, c\sigma_1), ..., (c_k, c\mu_k, c\sigma_k), bpk, bsk) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SIGN}}(gpk, csk)$
if $(bpk, bsk) \notin \mathcal{BK}$ **return** $\perp$
if $\exists i \in [1, k]$ s.t $\forall uid \in \mathsf{CUL} \cup \mathsf{HUL}$
$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) \neq \mathbf{upk}[uid]$
return $(c_i, c\mu_i, c\sigma_i, bpk, bsk)$
else return $\perp$

Figure 6.13: $\mathcal{A}'$ which breaks the traceability of commuting group signatures using $\mathcal{A}$ which breaks the `tier`-2 traceability of CLS–CGS

$[1, k]$ such that $\forall uid \in \mathsf{CUL} \cup \mathsf{HUL}$:

$$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) \neq \mathbf{upk}[uid]$$

then they will break the traceability of commuting group signatures.

Let $L' = |\{i \in [k] : \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) = \mathbf{upk}[uid] \text{ and } uid \in \mathsf{HUL}\}|$.

Let $C' = |\{i \in [k] : \mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) = \mathbf{upk}[uid] \text{ and } uid \in \mathsf{CUL}\}|$.

If $\mathcal{A}'$ aborts then $k \leq C' + L'$. However if $\mathcal{A}$ is successful, then $k > \mathsf{CUL} + L$, where $L = |\{uid \in \mathsf{HUL} : \exists i \text{ s.t } (uid, \mathsf{CGS.UnblindM}(bsk, c_i), bpk) \in \mathsf{SL}\}|$.

Therefore, either $C' > \mathsf{CUL}$ or $L' > L$.

If $C' > \mathsf{CUL}$, two signatures both open to the same corrupted user. Therefore for some $(i, j) \in [k]^2$, $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) =$

$\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_j, c\mu_j, c\sigma_j))$. However as the two signatures are unlinked we have:

$$\mathsf{CGS.UnblindUser}(bsk, e(g_1, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i))^r) \neq$$

$$\mathsf{CGS.UnblindUser}(bsk, e(g_1, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_j, c\mu_j, c\sigma_j)^r),$$

which is a contradiction.

If $L' > L$, let $L'' = |\{uid \in \mathsf{HUL} : \exists i \in [k]$ s.t $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) = \mathbf{upk}[uid]\}|$, as no two signatures will open to the same honest user, due to the same argument, $L' = L''$, and so $L'' > L$. Therefore there exists an honest user uid^* and $i \in [k]$ such that $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, bpk, csk_1, c_i, c\mu_i, c\sigma_i)) = \mathbf{upk}[uid^*]$, but $(uid^*, \mathsf{CGS.UnblindM}(bsk, c_i), bpk) \notin \mathsf{SL}$. In this case, $\mathcal{A}$ could break the non–frameability of commuting group signatures.

Therefore $\mathcal{A}'$ aborts with negligible probability, and so assuming the traceability and non–frameability of commuting group signatures, our CLS–CGS construction satisfies `tier`-2 traceability.

We now show that our CLS–CGS construction satisfies `tier`-1 traceability. Clearly if an adversary could break `tier`-1 traceability they could also break `tier`-2 traceability by simply blinding the group signatures. This is due to the fact that blinded and standard signatures are commuting group signatures. Commuting group signatures satisfy the commutative behaviour requirement. If a valid signature is blinded, this will result in a valid blinded signature. As our construction satisfies `tier`-2 traceability, this is a contradiction, and so our construction must also satisfy `tier`-1 traceability.

□

6.5 Concrete Instantiation of CLS–CGS construction

We now show that the building blocks of our CLS–CGS construction can be instantiated, based on the *Asymmetric Double Hidden SDH* [65] and *SXDH* assumption. For this, we need that the commuting group signatures construction given in Section 5.4 and instan-

tiated in Section 5.6, satisfies the additional structural assumptions and extractability in Section 6.3, and that the proof of unblinding can also be instantiated.

Firstly, due to the automorphic signature scheme [65] used, $usk \in \mathbb{Z}_p^*$, and $upk = g_2^{usk}$. Outputs of $\mathsf{CGS.OpenBlind}$ are also elements of $\mathbb{G}_2$.

Our construction for commuting group signatures satisfies $e(g_1, \mathsf{CGS.UnblindUser}(bsk, cupk))^r = \mathsf{CGS.UnblindUser}(bsk, e(g_1, cupk)^r)$ because letting $cupk = (cupk_1, cupk_2)$:

$$e(g_1, \mathsf{CGS.UnblindUser}(bsk, cupk))^r = e(g_1, (cupk_2 cupk_1^{-bsk}))^r =$$

$$e(g_1, cupk_2^r) e(g_1, cupk_1^r)^{-bsk} = \mathsf{CGS.UnblindUser}(bsk, e(g_1, cupk)^r).$$

Our construction satisfies $\mathsf{CGS.BlindUser}(bpk, upk))^r = \mathsf{CGS.BlindUser}(bpk, upk^r)$ because:

$$\mathsf{CGS.BlindUser}(bpk, upk; r'))^r = (\hat{g}^{rr'}, upk^r bpk_2^{rr'}) = \mathsf{CGS.BlindUser}(bpk, upk^r; rr')),$$

and as r' is chosen uniformly and independently, then rr' is distributed identically.

6.5.1 Extractability

We now show that that our commuting group signature construction satisfies extractability, assuming the $\mathsf{cm\text{-}NIZK}$ satisfies controlled malleable simulation soundness. We present simulators S_1, S_2, S_3:

$S_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow\!\!\$\, \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow\!\!\$\, \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$

$g \leftarrow\!\!\$\, \mathbb{G}_1, \hat{g} \leftarrow\!\!\$\, \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s) \leftarrow\!\!\$\, S_{1,\mathsf{cm\text{-}NIZK}}(1^\tau)$

return $(((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}}), \tau_s)$

$S_2(\mathsf{param}, \mathsf{st}_{\mathsf{Join}}^{uid}, M_{\mathsf{in}}, \tau_s)$

Identical to Join except π_{join} is simulated

$S_3(\mathsf{param}, gpk, bpk, \mathbf{upk}[uid], m, \tau_s)$

$\alpha \leftarrow\!\!\$\ \mathbb{Z}_p^*, \mu \leftarrow (\hat{g}^\alpha, 1, \mathbf{upk}[uid] opk^\alpha), \beta \leftarrow 0, \gamma \leftarrow 0, c \leftarrow (1, m)$

$\sigma \leftarrow\!\!\$\ S_{2,\mathsf{cm\text{-}NIZK}}(\sigma_{\mathsf{crs}}, \tau_s, (opk, bpk, ipk, \mu, c))$ **return** (μ, σ)

Due to the zero–knowledge of the SPK and the cm-NIZK, the following advantage is negligible in τ:

$$\left|\Pr[\mathbf{Exp}^{sim-0}_{\mathcal{A},\Pi}(\tau) = 1] - \Pr[\mathbf{Exp}^{sim-1}_{\mathcal{A},\Pi}(\tau) = 1]\right|.$$

We now provide SE_1, E_2. As outputs of $SE_{1,\mathsf{cm\text{-}NIZK}}$ are distributed identically to outputs of $S_{1,\mathsf{cm\text{-}NIZK}}$, outputs of SE_1 are identically distributed to outputs of S_1.

$SE_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$

$\mathsf{param}_{\mathsf{auto}1} \leftarrow\!\!\$\ \mathsf{ASetup}_1(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)), \mathsf{param}_{\mathsf{auto}2} \leftarrow\!\!\$\ \mathsf{ASetup}_2(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2))$

$g \leftarrow\!\!\$\ \mathbb{G}_1, \hat{g} \leftarrow\!\!\$\ \mathbb{G}_2, (\sigma_{\mathsf{crs}}, \tau_s, \tau_s) \leftarrow\!\!\$\ SE_{1,\mathsf{cm\text{-}NIZK}}(1^\tau)$

return $(((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2), \mathsf{param}_{\mathsf{auto}1}, \mathsf{param}_{\mathsf{auto}2}, g, \hat{g}, \sigma_{\mathsf{crs}}), \tau_s, \tau_e)$

$E_2(gpk, bpk, isk, osk, bsk, c, c\mu, c\sigma, \tau_e)$

$(upk', \cdot, \cdot, \cdot, \cdot, ...) \leftarrow E_{2,\mathsf{cm\text{-}NIZK}}(\sigma_{\mathsf{crs}}, \tau_e, (opk, bpk, ipk, c\mu, c), c\sigma)$ **return** upk'

The extractor $E_{2,\mathsf{cm\text{-}NIZK}}$ will either successfully extract upk', or instead will extract a transformation T and statement $(opk, bpk, ipk, c\mu', c')$ such that $(opk, bpk, ipk, c\mu, c) = T_{\mathsf{inst}}(opk, bpk, ipk, c\mu', c')$. If it outputs the latter, $(c\mu', c')$ was input to $S_{2,\mathsf{cm\text{-}NIZK}}$, and $(c\mu', c')$ is a re-randomisation of $(c\mu, c)$, therefore uid^* was input to S_3, such that $\mathsf{CGS.UnblindUser}(bsk, \mathsf{CGS.OpenBlind}(gpk, osk, c, c\mu, c\sigma)) = \mathbf{upk}[uid^*]$; in which case the extractor will not need to extract upk'.

6.5.2 Instantiating the Proof of Unblinding

We show how to instantiate the proof of unblinding π_{ub}:

$$\pi_{\mathsf{ub}} = \mathsf{SPK}\{bsk : \overline{\mu} = \mathsf{CGS.UnblindUser}(bsk, \overline{c\mu}) \wedge m = \mathsf{CGS.UnblindM}(bsk, \overline{c}) \wedge$$

$$(bsk, bpk) \in \mathcal{BPK}\}$$

$$= \mathsf{SPK}\{bsk_1, bsk_2 : \overline{\mu} = \overline{c\mu}_2 \overline{c\mu}_1^{-bsk_2} \wedge m = \overline{c}_2 \overline{c}_1^{-bsk_1} \wedge bpk_1 = g^{bsk_1} \wedge bpk_2 = \hat{g}^{bsk_2}\}$$

For transforming interactive into non-interactive zero-knowledge proofs, we rely on the Fiat-Shamir heuristic that ensures security in the random oracle model.

6.5.3 Efficiency

We focus on the computational overhead, as this is the key barrier to overcome to allow for the practical implementation of this work. The `tier`-1 and `tier`-2 signatures are the size of standard and blinded commuting group signatures respectively. Therefore, both have size $328\mathbb{G}_1 + 314\mathbb{G}_2$. Our CLS+ construction is significantly less efficient than that of our CLS construction in Chapter 4, but demonstrates that the stronger CLS+ security model can be achieved and the assumption of trusted data lakes can be avoided.

6.6 Summary

In this chapter we propose convertibly linkable group signatures, an extension of group signatures with selective linkability given in Chapter 4. We allow for authentication to be preserved during convert queries, and remove the assumption that inputs to the converter are well formed. We have extended the CLS model to provide a formal security model for this primitive. We have given a provably secure construction in this model, making use of commuting group signatures from Chapter 5, and a standard signature scheme. We then show that our instantiation of commuting group signatures can be used to build a concrete instantiation. We finally analyse the efficiency in comparison to our CLS instantiation.

Chapter 7

Concluding Remarks

In this thesis we have introduced several variants of group signatures that have applications in both reputation systems and the collection and processing of data.

In Chapter 3 we introduced a new cryptographic model for reputation systems. Reputation values are based on a user's entire behaviour, whilst a user's activities cannot be linked together. We have shown how a construction satisfying this model can be built based on direct anonymous attestation and a variant of a group signature scheme that allows reputation to be bound to the signature. We prove that this construction is secure based on the DDH, q-SDH and LRSW assumptions, as well as the security of the signature proofs of knowledge used. We then show that the signature proofs of knowledge can be instantiated in the random oracle model, and that this instantiation comes at a reasonable additional efficiency cost in comparison to existing work.

In Chapter 4 we introduced group signatures with selective linkability, a new variant of group signatures with applications in privacy–preserving technologies. Signatures can be linked in a more flexible, controlled way than via an opener, as in standard group signatures. Signatures are unlinkable by default but can be obliviously converted into a linked representation by the converter. The linkage in non-transitive, i.e., only holds within a particular convert query, which ensures that a list of linked signatures cannot be built up over time. We formally define the security of this primitive, and provide a construction that provably satisfies this model. This construction makes use of BBS+ signatures, ElGamal encryption and signature proofs of knowledge. We then provide

proofs of security for this construction given the DDH and q-SDH assumptions, as well as the security of the signature proof of knowledge. We then show that the signature proofs of knowledge can be instantiated in the random oracle model, and that this instantiation is reasonably efficient.

In Chapter 5 we introduce commuting group signatures, a variant of group signatures that are later used as a building block in Chapter 6. We extend the existing commuting signatures primitive to group signatures, introducing group signatures that can be blinded, whilst remaining publicly verifiable. They can also still be opened, outputting a blinded user identity. We first provide a formal model, defining the syntax and security properties required for this new primitive. We then provide a construction based on controlled malleable proof protocols, automorphic signatures and signature proofs of knowledge. Controlled malleable proofs allow the proof to be transformed when signatures are blinded, whilst still ensuring unforgeability of the signatures because the malleability is controlled. We then prove this construction secure, given the security of the building blocks and the DDH assumption. We show that our building blocks can be instantiated given the ADHSDH and SXDH assumptions in the random oracle model. We must particularly ensure that the controlled malleable proof protocols can be instantiated for the relation used in signing and the transformation used in blinding. We finally provide the sizes of signatures in terms of group elements.

In Chapter 6 we extend and strengthen the primitive defined in Chapter 4. We consider a setting where the data lake, who queries unlinkable user data to the converter, and data processor, who receives linked data from the converter for processing, are separate entities. Previously, we assumed that these were the same entity or trusted each other fully. We no longer assume that the data lake provides the converter with well formed inputs. We also must preserve the authentication of data throughout the conversion, so that the data processor is assured that data originates from a valid group member. We first extend the previous security model, to take into account this new setting. We then provide a construction that makes use of commuting group signatures, defined in Chapter 5, as a building block, and a standard digital signature scheme. We prove that our scheme is secure given the DDH assumption, as well as the security of the building blocks. We finally show that these building blocks can be instantiated, given the ADHSDH and SXDH assumptions in the random oracle model, and provide efficiency figures. The efficiency is significantly worse than in Chapter 4, however we show that our new strengthened security

model is achievable.

There are several potential directions for further work within this thesis. Firstly, in Chapter 3 we provide a static model where users all join at the beginning of the scheme. Extending this model to the dynamic setting, where users can join throughout, would be an improvement. Both building blocks in our construction are in the dynamic model, and so it should be possible to adapt the construction.

Furthermore, in Chapter 3, we provide a concrete construction using the two building blocks, and prove this is secure. It would also be interesting to formally define the security of the group signatures bound to reputation. A modular construction could then be built, so that any secure DAA scheme and group signature scheme bound to reputation could be used to build a secure reputation system.

In both Chapter 4 and Chapter 6, compared with the anonymity requirements of conventional dynamic group signatures, our anonymity notions are somewhat weaker as we do not allow the adversary to corrupt the two challenge users after it received the challenge signature. This means that our privacy related requirements do not yield forward anonymity. Given the conversion functionality that is inherent in our setting, achieving this stronger notion seems challenging. In fact, for the related problem of group signatures with user-controlled linkability with signature-based revocation, forward anonymity has not been achieved by any of the existing schemes. It would be interesting to determine whether this stronger security is achievable in this setting. We could also consider extending both security models to allow for the revocation of users, as in the fully dynamic model for group signatures.

The online extractable signature proofs of knowledge used in Chapter 4 are costly in terms of efficiency. Therefore, if the construction could be built without such proofs, it would allow the work to be more easily implemented. Signatures for the scheme in Chapter 6 are significantly larger than for the scheme in Chapter 4. If a more efficient construction could be built, it would allow the strengthened primitive to be used in practise.

The security of all our constructions is based on hardness problems related to the discrete logarithm problem. This means they are all vulnerable to attacks by an adversary with quantum capabilities. A potential future direction could be to build these primitives from

post–quantum secure cryptography.

Bibliography

[1] Amazons third-party sellers ship record-breaking 2 billion items in 2014, but merchant numbers stay flat. https://techcrunch.com/2015/01/05/amazon-third-party-sellers-2014/. [Online; accessed 1st-April-2019].

[2] Eu general data protection regulation. https://gdpr-info.eu.

[3] Travis kalanick says uber has 40 million monthly active riders. https://techcrunch.com/2016/10/19/travis-kalanick-says-uber-has-40-million-monthly-active-riders/. [Online; accessed 1st-April-2019].

[4] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 418–433. Springer, Heidelberg, Apr. / May 2002.

[5] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236. Springer, Heidelberg, Aug. 2010.

[6] M. Abe, D. Hofheinz, R. Nishimaki, M. Ohkubo, and J. Pan. Compact structure-preserving signatures with almost tight security. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 548–580. Springer, Heidelberg, Aug. 2017.

[7] E. Androulaki, S. G. Choi, S. M. Bellovin, and T. Malkin. Reputation systems for anonymous networks. In *International Symposium on Privacy Enhancing Technologies*, pages 202–218. Springer, 2008.

[8] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. http://eprint.iacr.org/2005/385.

[9] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. In R. D. Prisco and M. Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 111–125. Springer, Heidelberg, Sept. 2006.

[10] L. Ballard, M. Green, B. de Medeiros, and F. Monrose. Correlation-resistant storage. Technical report, John Hopkins University, Computer Science Department # TR-SP-BGMM-050705. http://spar.isi.jhu.edu/~mgreen/correlation.pdf, 2005.

[11] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 26–45. Springer, Heidelberg, Aug. 1998.

[12] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.

[13] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.

[14] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153. Springer, Heidelberg, Feb. 2005.

[15] D. Bernhard, G. Fuchsbauer, E. Ghadafi, N. P. Smart, and B. Warinschi. Anonymous attestation with user-controlled linkability. *International Journal of Information Security*, 12(3):219–249, 2013.

[16] J. Bethencourt, E. Shi, and D. Song. Signatures of reputation. In R. Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 400–407. Springer, Heidelberg, Jan. 2010.

[17] P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi. Get shorty via group signatures without encryption. In J. A. Garay and R. D. Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 381–398. Springer, Heidelberg, Sept. 2010.

[18] J. Blömer, J. Juhnke, and C. Kolb. Anonymous and publicly linkable reputation systems. In R. Böhme and T. Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 478–488. Springer, Heidelberg, Jan. 2015.

[19] D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Heidelberg, May 2004.

[20] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, Apr. 2008.

[21] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, Aug. 2004.

[22] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.

[23] D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In V. Atluri, B. Pfitzmann, and P. McDaniel, editors, *ACM CCS 2004*, pages 168–177. ACM Press, Oct. 2004.

[24] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, and J. Groth. Foundations of fully dynamic group signatures. In M. Manulis, A.-R. Sadeghi, and S. Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 117–136. Springer, Heidelberg, June 2016.

[25] X. Boyen and B. Waters. Compact group signatures without random oracles. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 427–444. Springer, Heidelberg, May / June 2006.

[26] E. Bresson and J. Stern. Efficient revocation in group signatures. In K. Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 190–206. Springer, Heidelberg, Feb. 2001.

[27] E. Brickell. An efficient protocol for anonymously providing assurance of the container of a private key. *Submitted to the Trusted Computing Group*, 2003.

[28] E. Brickell and J. Li. Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 21–30, 2007.

[29] E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In V. Atluri, B. Pfitzmann, and P. McDaniel, editors, *ACM CCS 2004*, pages 132–145. ACM Press, Oct. 2004.

[30] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, and R. Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy, SP*, pages 901–920. IEEE, 2017.

[31] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven. The wonderful world of global random oracles. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, Apr. / May 2018.

[32] J. Camenisch, M. Drijvers, and A. Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *International Conference on Trust and Trustworthy Computing*, pages 1–20. Springer, 2016.

[33] J. Camenisch, M. Drijvers, and A. Lehmann. Universally composable direct anonymous attestation. In C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 234–264. Springer, Heidelberg, Mar. 2016.

[34] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In C. Blundo and S. Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 120–133. Springer, Heidelberg, Sept. 2005.

[35] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized Schnorr proofs. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 425–442. Springer, Heidelberg, Apr. 2009.

[36] J. Camenisch and A. Lehmann. (Un)linkable pseudonyms for governmental databases. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 1467–1479. ACM Press, Oct. 2015.

[37] J. Camenisch and A. Lehmann. Privacy-preserving user-auditable pseudonym systems. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 269–284. IEEE, 2017.

[38] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, Aug. 2002.

[39] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, Aug. 2004.

[40] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, Aug. 2003.

[41] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In B. S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Heidelberg, Aug. 1997.

[42] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited (preliminary version). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 209–218, New York, NY, USA, 1998. ACM.

[43] R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing chosen-ciphertext security. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, Heidelberg, Aug. 2003.

[44] M. Chase and M. Kohlweiss. A domain transformation for structure-preserving signatures on group elements. Cryptology ePrint Archive, Report 2011/342, 2011. `http://eprint.iacr.org/2011/342`.

[45] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, Heidelberg, Apr. 2012.

[46] M. Chase and A. Lysyanskaya. On signatures of knowledge. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, Aug. 2006.

[47] S. Chatterjee and A. Menezes. On cryptographic protocols employing asymmetric pairings the role of *psi* revisited. *Discrete Applied Mathematics*, 159(13):1311–1322, 2011.

[48] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

[49] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, Apr. 1991.

[50] C.-K. Chu, J. K. Liu, X. Huang, and J. Zhou. Verifier-local revocation group signatures with time-bound keys. In H. Y. Youm and Y. Won, editors, *ASIACCS 12*, pages 26–27. ACM Press, May 2012.

[51] M. R. Clark, K. Stewart, and K. M. Hopkinson. Dynamic, privacy-preserving decentralized reputation systems. *IEEE Transactions on Mobile Computing*, 16(9):2506–2517, 2017.

[52] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, Aug. 2001.

[53] C. Delerablée and D. Pointcheval. Dynamic fully anonymous short group signatures. In *Progress in Cryptology-VIETCRYPT 2006*, pages 193–210. Springer, 2006.

[54] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 150–157. ACM, 2000.

[55] X. Ding, G. Tsudik, and S. Xu. Leak-free group signatures with immediate revocation. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 608–615. IEEE, 2004.

[56] X. Ding, G. Tsudik, and S. Xu. Leak-free mediated group signatures. *Journal of Computer Security*, 17(4):489–514, 2009.

[57] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 609–626. Springer, Heidelberg, May 2004.

[58] A. El Kaafarani, S. Katsumata, and R. Solomon. Anonymous reputation systems achieving full dynamicity from lattices. In *Twenty-Second International Conference on Financial Cryptography and Data Security*, forthcoming.

[59] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, Aug. 1984.

[60] T. ElGamal. On computing logarithms over finite fields. In H. C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 396–402. Springer, Heidelberg, Aug. 1986.

[61] K. Emura, T. Hayashi, and A. Ishida. Group signatures with time-bound keys revisited: A new model and an efficient construction. In R. Karri, O. Sinanoglu, A.-R. Sadeghi, and X. Yi, editors, *ASIACCS 17*, pages 777–788. ACM Press, Apr. 2017.

[62] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. On the non-malleability of the Fiat-Shamir transform. In S. D. Galbraith and M. Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, Dec. 2012.

[63] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, Aug. 1987.

[64] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, Aug. 2005.

[65] G. Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. Cryptology ePrint Archive, Report 2009/320, 2009. `http://eprint.iacr.org/2009/320`.

[66] G. Fuchsbauer. Commuting signatures and verifiable encryption. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 224–245. Springer, Heidelberg, May 2011.

[67] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[68] D. Galindo and E. R. Verheul. Microdata sharing via pseudonymization. *Joint UNECE/Eurostat work session on statistical data confidentiality*, 2007.

[69] L. Garms and A. Lehmann. Group signatures with selective linkability. In D. Lin and K. Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 190–220. Springer, Heidelberg, Apr. 2019.

[70] L. Garms, K. Martin, and S.-L. Ng. Reputation schemes for pervasive social networks with anonymity. In *Proceedings of the fifteenth International Conference on Privacy, Security and Trust (PST 2017), IEEE*, 2017.

[71] L. Garms and E. A. Quaglia. A new approach to modelling centralised reputation systems. In J. Buchmann, A. Nitaj, and T. eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 429–447. Springer, Heidelberg, July 2019.

[72] E. Ghadafi. Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability. In D. F. Aranha and A. Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 327–347. Springer, Heidelberg, Sept. 2015.

[73] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[74] S. D. Gordon, J. Katz, and V. Vaikuntanathan. A group signature scheme from lattice assumptions. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 395–412. Springer, Heidelberg, Dec. 2010.

[75] J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In X. Lai and K. Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Heidelberg, Dec. 2006.

[76] J. Groth. Fully anonymous group signatures without random oracles. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 164–180. Springer, Heidelberg, Dec. 2007.

[77] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, Apr. 2008.

[78] O. Hasan, L. Brunie, and E. Bertino. Preserving privacy of feedback providers in decentralized reputation systems. *Computers & Security*, 31(7):816 – 826, 2012.

[79] J. Y. Hwang, S. Lee, B.-H. Chung, H. S. Cho, and D. Nyang. Short group signatures with controllable linkability. In *Lightweight Security & Privacy: Devices, Protocols and Applications (LightSec), 2011 Workshop on*, pages 44–52. IEEE, 2011.

[80] J. Y. Hwang, S. Lee, B.-H. Chung, H. S. Cho, and D. Nyang. Group signatures with controllable linkability for dynamic membership. *Information Sciences*, 222:761–778, 2013.

[81] A. Ishida, K. Emura, G. Hanaoka, Y. Sakai, and K. Tanaka. Group signature with deniability: How to disavow a signature. In S. Foresti and G. Persiano, editors, *CANS 16*, volume 10052 of *LNCS*, pages 228–244. Springer, Heidelberg, Nov. 2016.

[82] J. Katz and Y. Lindell. *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.

[83] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 571–589. Springer, Heidelberg, May 2004.

[84] M. Kohlweiss and I. Miers. Accountable metadata-hiding escrow: A group signature case study. *PoPETs*, 2015(2):206–221, Apr. 2015.

[85] S. Krenn, K. Samelin, and C. Striecks. Practical group-signatures with privacy-friendly openings. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, pages 10:1–10:10. ACM, 2019.

[86] V. Kumar, H. Li, J.-M. J. Park, K. Bian, and Y. Yang. Group signatures with probabilistic revocation: A computationally-scalable approach for providing privacy-preserving authentication. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 1334–1345. ACM Press, Oct. 2015.

[87] F. Laguillaumie, A. Langlois, B. Libert, and D. Stehlé. Lattice-based group signatures with logarithmic signature size. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 41–61. Springer, Heidelberg, Dec. 2013.

[88] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 373–403. Springer, Heidelberg, Dec. 2016.

[89] B. Libert, S. Ling, K. Nguyen, and H. Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 1–31. Springer, Heidelberg, May 2016.

[90] B. Libert, T. Peters, and M. Yung. Group signatures with almost-for-free revocation. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 571–589. Springer, Heidelberg, Aug. 2012.

[91] B. Libert, T. Peters, and M. Yung. Scalable group signatures with revocation. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 609–627. Springer, Heidelberg, Apr. 2012.

[92] S. Ling, K. Nguyen, and H. Wang. Group signatures from lattices: Simpler, tighter, shorter, ring-based. In J. Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 427–449. Springer, Heidelberg, Mar. / Apr. 2015.

[93] S. Ling, K. Nguyen, H. Wang, and Y. Xu. Lattice-based group signatures: Achieving full dynamicity with ease. In D. Gollmann, A. Miyaji, and H. Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 293–312. Springer, Heidelberg, July 2017.

[94] J. K. Liu, V. K. Wei, and D. S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Australasian Conference on Information Security and Privacy*, pages 325–335. Springer, 2004.

[95] Z. Liu, S. S. Yau, D. Peng, and Y. Yin. A flexible trust model for distributed service infrastructures. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 108–115. IEEE, 2008.

[96] A. Lysyanskaya and Z. Ramzan. Group blind digital signatures: A scalable solution to electronic cash. In R. Hirschfeld, editor, *FC'98*, volume 1465 of *LNCS*, pages 184–197. Springer, Heidelberg, Feb. 1998.

[97] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In H. M. Heys and C. M. Adams, editors, *SAC 1999*, volume 1758 of *LNCS*, pages 184–199. Springer, Heidelberg, Aug. 1999.

[98] M. Manulis. Democratic group signatures: On an example of joint ventures (fast abstract). In F.-C. Lin, D.-T. Lee, B.-S. Lin, S. Shieh, and S. Jajodia, editors, *ASIACCS 06*, page 365. ACM Press, Mar. 2006.

[99] M. Manulis, N. Fleischhacker, F. Günther, F. Kiefer, and B. Poettrering. Group signatures: Authentication with privacy. *Bundesamt fur Sicherheit in der Informationstechnik, Bonn, Germany, Tech. Rep*, 2012.

[100] M. Manulis, A.-R. Sadeghi, and J. Schwenk. Linkable democratic group signatures. In *International Conference on Information Security Practice and Experience*, pages 187–201. Springer, 2006.

[101] F. G. Mármol and G. M. Pérez. Security threats scenarios in trust and reputation models for distributed systems. *Computers & Security*, 28(7):545–556, 2009.

[102] S. Marti and H. Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472–484, 2006.

[103] T. Nakanishi, H. Fujii, Y. Hira, and N. Funabiki. Revocable group signature schemes with constant costs for signing and verifying. In S. Jarecki and G. Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 463–480. Springer, Heidelberg, Mar. 2009.

[104] T. Nakanishi, Y. Hira, and N. Funabiki. Forward-secure group signatures from pairings. In H. Shacham and B. Waters, editors, *PAIRING 2009*, volume 5671 of *LNCS*, pages 171–186. Springer, Heidelberg, Aug. 2009.

[105] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, Oct. 1997.

[106] S.-L. Ng, K. Martin, L. Chen, and Q. Li. Private reputation retrieval in public - a privacy-aware announcement scheme for vanets. *IET Information Security, DOI: 10.1049/iet-ifs.2014.0316*, 2016.

[107] L. Nguyen. Accumulators from bilinear pairings and applications. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer, Heidelberg, Feb. 2005.

[108] T. Ono and K. Yoneyama. On randomness exposure resilience of group signatures. *IEICE TRANSACTIONS on Information and Systems*, 100(10):2357–2367, 2017.

[109] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[110] R. Pass. On deniability in the common reference string and random oracle model. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, Heidelberg, Aug. 2003.

[111] E. Pavlov, J. S. Rosenschein, and Z. Topol. Supporting privacy in decentralized additive reputation systems. In *International Conference on Trust Management*, pages 108–119. Springer, 2004.

[112] R. Petrlic, S. Lutters, and C. Sorge. Privacy-preserving reputation management. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1712–1718, New York, NY, USA, 2014. ACM.

[113] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

[114] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

[115] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Heidelberg, Dec. 2001.

[116] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, Oct. 1999.

[117] Y. Sakai, K. Emura, G. Hanaoka, Y. Kawai, T. Matsuda, and K. Omote. Group signatures with message-dependent opening. In M. Abdalla and T. Lange, editors, *PAIRING 2012*, volume 7708 of *LNCS*, pages 270–294. Springer, Heidelberg, May 2013.

[118] Y. Sakai, J. C. N. Schuldt, K. Emura, G. Hanaoka, and K. Ohta. On the security of dynamic group signatures: Preventing signature hijacking. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 715–732. Springer, Heidelberg, May 2012.

[119] M. Scott. Pairing implementation revisited. Cryptology ePrint Archive, Report 2019/077, 2019. https://eprint.iacr.org/2019/077.

[120] D. Slamanig, R. Spreitzer, and T. Unterluggauer. Adding controllable linkability to pairing-based group signatures for free. In *International Conference on Information Security*, pages 388–400. Springer, 2014.

[121] D. X. Song. Practical forward secure group signature schemes. In M. K. Reiter and P. Samarati, editors, *ACM CCS 2001*, pages 225–234. ACM Press, Nov. 2001.

[122] M. Stadler. Publicly verifiable secret sharing. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 190–199. Springer, Heidelberg, May 1996.

[123] G. Traverso, D. Butin, J. Buchmann, and A. Palesandro. Coalition-resistant peer rating for long-term confidentiality. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pages 1–10. IEEE, 2018.

[124] G. Tsudik and S. Xu. Accumulating composites and improved group signing. In C.-S. Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 269–286. Springer, Heidelberg, Nov. / Dec. 2003.

[125] Z. Yan and Y. Chen. Adcontrep: a privacy enhanced reputation system for manet content services. In *International Conference on Ubiquitous Intelligence and Computing*, pages 414–429. Springer, 2010.

[126] Z. Yan, Y. Chen, and Y. Shen. A practical reputation system for pervasive social chatting. *Journal of Computer and System Sciences*, 79(5):556–572, 2013.

[127] A. L. Young and M. Yung. Semantically secure anonymity: Foundations of reencryption. In D. Catalano and R. De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 255–273. Springer, Heidelberg, Sept. 2018.

[128] E. Zhai, D. I. Wolinsky, R. Chen, E. Syta, C. Teng, and B. Ford. Anonrep: towards tracking-resistant anonymous reputation. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 583–596. USENIX Association, 2016.

Appendix A

Additional Security Proofs for our RS-GS construction

A.1 Traceability

Assuming the random oracle model, the SPK is zero–knowledge and simulation sound extractable, and the q-SDH assumption, our RS-GS construction satisfies traceability.

Proof. We show that if an adversary $\mathcal{A}'$ exists, such that $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1] = \epsilon$, for some $\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}$ with $|\mathcal{R}|$ polynomial in τ, $n = |\mathcal{U}|$ polynomial in τ, there are l different values of (r, t) queried to the $\mathcal{H}$ oracle, or ALLREP oracle, and ϵ is non–negligible in τ, then we can can build an adversary $\mathcal{A}$, that solves the q-SDH problem, where $q = n$, in polynomial–time. We describe $\mathcal{A}$ in Figures A.1 and A.2. We then describe why the simulation given in Figures A.1 and A.2 and the traceability experiment are indistinguishable to $\mathcal{A}$, and how $\mathcal{A}'$ works.

We first show that all inputs that $\mathcal{A}$ provides to $\mathcal{A}'$ are distributed identically to the traceability experiment.

Simulating $(gpk, osk, \mathcal{ID})$**.** $W = T_2^{\gamma} = G_2^{\gamma}$. G_1 is chosen randomly due to μ being chosen randomly and independently. As ν_1 is chosen randomly and independently, H is

$\mathsf{USK}(uid)$

if $b = 1$ and $uid = uid^*$ $\quad \mathcal{A}$ aborts $\quad$ **else** $\quad$ **return** $\mathbf{usk}[uid]$

$\mathsf{POST}(I, uid, r, t, \omega)$

if $b = 1$ and $uid = uid^*$
  if $\hat{t}(\omega, WG_2^{x_{uid^*}}) \neq \hat{t}(\mathcal{H}(r,t), G_2)$ **return** $\perp$
  $\rho_1, \rho_2 \leftarrow\!\!\$\, \mathbb{Z}_p, T_1 \leftarrow K^{\rho_1}, T_2 \leftarrow Z_{uid^*}\omega H^{\rho_1}, T_3 \leftarrow K^{\rho_2}, T_4 \leftarrow Z_{uid^*}\omega G^{\rho_2}$
  $\tilde{G_1} \leftarrow \mathcal{H}(r,t) \cdot G_1$, Simulate π with $T_1, T_2, T_3, T_4, \tilde{G_1}$
  return $\Omega \leftarrow (T_1, T_2, T_3, T_4, \pi)$
else return $\mathsf{PostItem}(gpk, I, \mathbf{usk}[uid], r, t, \omega)$

$\mathsf{SENDFB}(uid, fb, (I, r, t, \Omega))$

$\Phi \leftarrow\!\!\$\, \mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb)$

$\mathsf{ALLREP}(uid, t, r)$

if $(uid, t, r, \mathsf{out}) \in AR$, **return** $(\mathsf{out}, r, \mathcal{ID})$

$\mathsf{out}' \leftarrow \mathcal{H}(r,t)$, let $((r,t), \mathsf{out}', \chi) \in HL, \mathsf{out}'' \leftarrow \prod_{j=0}^{n-1} (T_1^{\gamma^j})^{\chi\kappa_{uid,j}}$

$AR \leftarrow AR \cup (uid, t, r, \mathsf{out}''), \mathcal{ID} \leftarrow \mathcal{ID} \cup (uid, r, t, Z_{uid}\mathsf{out}'')$, $\quad$ **return** $(\mathsf{out}'', r, \mathcal{ID})$

$\mathcal{H}(\mathsf{in})$

if $\exists(\mathsf{in}, \mathsf{out}) \in HL$ $\quad$ **return** out

else $\chi \leftarrow\!\!\$\, \mathbb{Z}_p^*, \mathsf{out} \leftarrow \prod_{j=0}^{n} (T_1^{\gamma^j})^{\chi\lambda_j}$, add $(\mathsf{in}, \mathsf{out}, \chi)$ to HL, $\quad$ **return** out

Figure A.1: Simulated answers to oracle queries for our traceability proof

$\mathcal{A}(T_1, T_1^{\gamma}, T_1^{\gamma^2}, ..., T_1^{\gamma^q}, T_2, T_2^{\gamma})$

Create Empty Lists $HL, AR, b \leftarrow\$ \{0,1\}, uid' \leftarrow\$ \mathcal{U}$

if $b = 0, \mathcal{V} \leftarrow \mathcal{U}, n' \leftarrow n$

if $b = 1, uid^* \leftarrow\$ \mathcal{U}\backslash\{uid'\}, \mathcal{V} \leftarrow \mathcal{U}\backslash\{uid^*\}, n' \leftarrow n-1, x_{uid^*} \leftarrow\$ \mathbb{Z}_p^*, Z_{uid^*} \leftarrow\$ \mathbb{G}_1$

$\forall uid \in \mathcal{V}\backslash\{uid'\}, x_{uid}, y_{uid} \leftarrow\$ \mathbb{Z}_p^*$

$$\mu \leftarrow\$ \mathbb{Z}_p^*, \text{let } f(X) = \prod_{uid \in \mathcal{V}\backslash\{uid'\}} (X + x_{uid}) = \sum_{i=1}^{n'-1} \zeta_i X^i; G_1 \leftarrow \prod_{i=0}^{n'-1} (T_1^{\gamma^i})^{\mu\zeta_i}, \Gamma \leftarrow \prod_{i=0}^{n'-1} (T_1^{\gamma^{(i+1)}})^{\mu\zeta_i}$$

$G_2 \leftarrow T_2, W \leftarrow T_2^{\gamma}, x, \nu_1, \nu_2 \leftarrow\$ \mathbb{Z}_p^*, H \leftarrow ((\Gamma G_1^x)^{\nu_1} G_1^{-1})^{1/\nu_2}$

$\xi_1 \leftarrow\$ \mathbb{Z}_p^*, K \leftarrow H^{1/\xi_1}, \xi_2 \leftarrow\$ \mathbb{Z}_p^*, G \leftarrow K^{\xi_2}, gpk_1 \leftarrow (G_1, K, H, G, G_2, W)$

$$\forall uid \in \mathcal{V}\backslash\{uid'\} \text{ let } f_{uid}(X) = \prod_{\hat{uid} \in \mathcal{V}\backslash\{uid', uid\}} (X + x_{\hat{uid}}) = \sum_{j=0}^{n'-2} \eta_{uid,j} X^j; B_{uid} \leftarrow \prod_{j=0}^{n'-2} (T_1^{\gamma^j})^{\mu\eta_{uid,j}}$$

$$Z_{uid} \leftarrow B_{uid}(B_{uid}^{((x-x_{uid})\nu_1 - 1)/\nu_2} G_1^{\nu_1/\nu_2})^{y_{uid}}, \mathbf{usk}_1[uid] \leftarrow (Z_{uid}, x_{uid}, y_{uid})$$

$y_{uid'} \leftarrow \nu_2, x_{uid'} \leftarrow x, Z_{uid'} \leftarrow G_1^{\nu_1}, \mathbf{usk}_1[uid'] \leftarrow (Z_{uid'}, x_{uid'}, y_{uid'})$

$$\text{Let } g(X) = \prod_{uid \in \mathcal{U}} (X + x_{uid}) = \sum_{j=0}^{n} \lambda_j X^j$$

$$\forall uid \in \mathcal{U}, \text{set } g_{uid}(X) = g(X)/(X + x_{uid}) = \sum_{j=0}^{n-1} \kappa_{uid,j} X^j$$

Finish computing $(\mathbf{usk}_2[uid], gpk_2, \mathcal{ID})$ as in Setup

$(I, r, t, \Omega, fb, \Phi, \mathbf{r}, \mathcal{L}, \mathcal{F}) \leftarrow\$ \mathcal{A}'^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP},\mathcal{H}}(gpk, osk, \mathcal{ID})$

Let $\Omega = (T_1, T_2, T_3, T_4, \pi)$

if $\nexists$out such that $((r,t), \mathsf{out}, \chi) \in HL$ **return** $\perp$

Extract $x^\star, \rho_1^\star, z^\star$ from $\pi, y^\star \leftarrow z^\star - x^\star \rho_1^\star, Q^\star \leftarrow T_2 T_1^{-\xi_1}$

if $x^\star = x_{uid}$ with $uid \in \mathcal{U}$ $\omega \leftarrow\$ \mathsf{ALLREP}(uid, t, r), \tilde{Q}^\star \leftarrow Q^\star \omega^{-1} = (G_1, H^{y^\star})^{1/(x^\star+\gamma)}$

  if $uid \notin \mathcal{C}$ **if** $x^\star \neq x_{uid^*}$ **return** $\perp$

    return $(\tilde{Q}^\star G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{\nu_2 - y^\star - \nu_1 y^\star (x^\star - x)}}, x^\star)$

  if $uid \in \mathcal{C}$ **if** $x^\star \neq x_{uid'}$ **return** $\perp$

    return $(\tilde{Q}^\star G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{\nu_2 - y^\star}}, x^\star)$

else return $(Q^\star G_1^{-\nu_1 y^\star/\nu_2} G_1^{-\chi/\mu})^{\frac{\nu_2}{\nu_2 - y^\star - (\nu_1 y^\star + \chi\nu_2/\mu)(x^\star - x)}}, x^\star)$

Figure A.2: $\mathcal{A}$ which solves the q-SDH problem, using $\mathcal{A}'$ which breaks traceability for the RS-GS construction

independent of G_1. $(\xi_1, \xi_2, K, G, \mathcal{ID})$ are chosen as in Setup. Therefore $(gpk, osk, \mathcal{ID})$ are distributed identically to the traceability experiment.

We will use the fact that $\Gamma = G_1^{\gamma}$ and $B_{uid} = G_1^{1/(\gamma + x_{uid})}$ later. This is because $G_1 = \prod_{i=0}^{n'-1} (T_1^{\gamma^i})^{\mu\zeta_i} = T_1^{\mu f(\gamma)}$, and so $\Gamma = \prod_{i=0}^{n'-1} (T_1^{\gamma^{(i+1)}})^{\mu\zeta_i} = \prod_{i=0}^{n'-1} T_1^{\gamma^i \mu\zeta_i \gamma} = G_1^{\gamma}$, and $B_{uid} = \prod_{j=0}^{n'-2} (T_1^{\gamma^j})^{\mu\eta_{uid,j}} = T_1^{\mu f_{uid}(\gamma)} = G_1^{1/(\gamma + x_{uid})}$.

Simulating the USK oracle. The USK oracle is distributed identically to the traceability experiment because, provided the oracle does not abort, if $uid \neq uid'$,

$$Z_{uid} = B_{uid}(B_{uid}^{((x-x_{uid})\nu_1 - 1)/\nu_2} G_1^{\nu_1/\nu_2})^{y_{uid}} = B_{uid}(B_{uid}^{((x-x_{uid})\nu_1 - 1)/\nu_2} B_{uid}^{\nu_1(\gamma + x_{uid})/\nu_2})^{y_{uid}}$$

$$= B_{uid} B_{uid}^{y_{uid}(\nu_1(\gamma+x)-1)/\nu_2} = (G_1 G_1^{y_{uid}(\nu_1(\gamma+x)-1)/\nu_2})^{1/(\gamma + x_{uid})} = (G_1 H^{y_{uid}})^{1/(\gamma + x_{uid})},$$

and $Z_{uid'} = G_1^{\nu_1} = (G_1 G_1^{\nu_1(\gamma+x)-1})^{1/(\gamma+x)} = (G_1 H^{y_{uid'}})^{1/(\gamma + x_{uid'})}$.

Simulating the POST oracle. If $b = 1$, and $uid = uid^*$ is input to the POST oracle, because Z_{uid^*} was chosen randomly and independently, it is distributed identically to the traceability experiment. The SPK can then be simulated due to the zero–knowledge property. The signature output is then distributed identically to in PostItem. If $b = 0$ or $uid \neq uid^*$, our oracle works in exactly the same way as in the traceability experiment.

Simulating all other oracles. The SENDFB oracle is identical to the traceability experiment. The ALLREP oracle is distributed identically to the traceability experiment, because $\prod_{j=0}^{n-1} (T_1^{\gamma^j})^{\chi\kappa_{uid,j}} = T_1^{\chi g(\gamma)/(\gamma + x_{uid})} = \mathcal{H}(r,t)^{1/(\gamma + x_{uid})}$. The hash oracle is distributed identically to the traceability experiment in the random oracle model, because χ is chosen randomly and independently every hash query.

Reduction to q-SDH. If $\mathcal{A}'$ is successful and outputs (I, r, t, Ω) then there exists out such that $((r,t), \mathsf{out}, \chi) \in HL$, and so $\mathcal{A}$ does not abort for this reason. This is because for $\mathcal{A}'$ to have output a valid signature on (r,t), they must have queried this to the hash oracle. As Ω was not output by the POST oracle, we can extract $(Q^\star, x^\star, y^\star)$ such that $Q^\star = (G_1, H^{y^\star} \mathsf{out})^{1/(x^\star + \gamma)}$, because Ω is a valid signature.

Assume $\mathcal{A}'$ is successful. If $x^\star = x_{uid}$ for $uid \in \mathcal{U}\backslash\mathcal{C}$, then we assume we chose $b = 1$ with probability $1/2$, and $uid = uid^*$ with probability $1/n$. $\mathcal{A}$ will not have aborted during USK, because uid^* was not corrupted. If $\nu_2 - y^\star - \nu_1 y^\star(x^\star - x) = 0$, then $\nu_1 = \frac{\nu_2 - y^\star}{y^\star(x^\star - x)}$. If $y^\star = 0$ then $\nu_2 = 0$, which is not possible. Therefore the adversary can obtain ν_1 and so break the discrete logarithm problem, which is implied by the q-SDH problem.

Due to the fact that $\tilde{Q}^\star = (G_1 H^{y^\star})^{1/(x^\star+\gamma)}$, then

$$(\tilde{Q}^\star G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{\nu_2 - y^\star - \nu_1 y^\star(x^\star - x)}} = (G_1 H^{y^\star})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}} G_1^{\frac{-\nu_1 y^\star}{\nu_2 - y^\star - \nu_1 y^\star(x^\star - x)}}$$

$$= (G_1 G_1^{y^\star(\nu_1(\gamma+x)-1)/\nu_2})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}} G_1^{\frac{-\nu_1 y^\star}{\nu_2 - y^\star - \nu_1 y^\star(x^\star - x)}}$$

$$= G_1^{\frac{y^\star\nu_1(\gamma+x) - y^\star + \nu_2 - y^\star\nu_1(\gamma+x^\star)}{(\gamma+x^\star)(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}} = G_1^{1/(\gamma+x^\star)}.$$

Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20]. Therefore, in this case, assuming $\mathcal{A}'$ was successful, $\mathcal{A}$ succeeds with probability $1/2n$.

If $x^\star = x_{uid}$ for some $uid \in \mathcal{C}$, then as $(G_1 H^{y_{uid}}\mathsf{out})^{1/(x_{uid}+\gamma)}$ is saved in $\mathcal{ID}$, $y^\star \neq y_{uid}$. $\tilde{Q}^\star = (G_1 H^{y^\star})^{1/(x^\star+\gamma)}$. Assuming $b = 0$ was chosen with probability $1/2$, the USK oracle does not abort. Therefore, assuming $x^\star = x_{uid'}$ with probability $1/n$, then $\mathcal{A}$ will not abort. As $\nu_2 = y_{uid'} \neq y^\star$, then $\nu_2 - y^\star \neq 0$.

Due to the fact that $\tilde{Q}^\star = (G_1 H^{y^\star})^{1/(x^\star+\gamma)}$, then

$$(\tilde{Q}^\star G_1^{-\nu_1 y^\star/\nu_2})^{\frac{\nu_2}{\nu_2 - y^\star}} = (G_1 H^{y^\star})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2 - y^\star)}} G_1^{\frac{-\nu_1 y^\star}{\nu_2 - y^\star}}$$

$$= (G_1 G_1^{y^\star(\nu_1(\gamma+x)-1)/\nu_2})^{\frac{\nu_2}{(\gamma+x^\star)(\nu_2 - y^\star)}} G_1^{\frac{-\nu_1 y^\star}{\nu_2 - y^\star}} = G_1^{\frac{y^\star\nu_1(\gamma+x) - y^\star + \nu_2 - y^\star\nu_1(\gamma+x^\star)}{(\gamma+x^\star)(\nu_2 - y^\star)}} = G_1^{1/(\gamma+x^\star)}.$$

Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20] . Therefore, in this case, assuming $\mathcal{A}'$ was successful, $\mathcal{A}$ succeeds with probability $1/2n$.

If $x^\star \neq x_{uid}$ with $uid \in \mathcal{U}$, then we assume $b = 0$ was chosen with probability $1/2$. Therefore, the USK oracle does not abort, and so $\mathcal{A}$ will not abort.

If $\nu_2 - y^\star - (\nu_1 y^\star + \chi\nu_2/\mu)(x^\star - x) = 0$, then $\chi = \frac{\mu(\nu_2 - y^\star - \nu_1 y^\star(x^\star - x))}{\nu_2(x^\star - x)}$. Therefore the

adversary can obtain χ and so break the discrete logarithm problem, which is implied by the q-SDH problem.

Due to the fact that,

$$\mathsf{out}^{1/(\gamma+x^\star)} = (\prod_{j=0}^{n}(T_1^{\gamma^j})^{\chi\lambda_j})^{1/(\gamma+x^\star)} = T_1^{\chi g(\gamma)/(\gamma+x^\star)} = G_1^{\frac{\chi(\gamma+x)}{\mu(\gamma+x^\star)}}$$

$$= \Gamma^{\chi/\mu(\gamma+x^\star)} G_1^{\chi x/\mu(\gamma+x^\star)} = (\Gamma^{\chi/\mu} G_1^{\chi x^\star/\mu})^{1/(\gamma+x^\star)} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}} = G_1^{\chi/\mu} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}},$$

and $Q^\star = (G_1 H^{y^\star}\mathsf{out})^{1/(\gamma+x^\star)}$,

and

$$(G_1 H^{y^\star})^{1/(\gamma+x^\star)} = (G_1 G_1^{y^\star(\nu_1(\gamma+x)-1)/\nu_2})^{1/(\gamma+x^\star)}$$

$$= G_1^{\frac{y^\star\nu_1 x^\star}{\nu_2(\gamma+x^\star)}} G_1^{\frac{y^\star\nu_1\gamma}{\nu_2(\gamma+x^\star)}} (G_1 G_1^{y^\star(\nu_1(x-x^\star)-1)/\nu_2})^{1/(\gamma+x^\star)} = G_1^{y^\star\nu_1/\nu_2} G_1^{\frac{\nu_2+y^\star(\nu_1(x-x^\star)-1)}{\nu_2(\gamma+x^\star)}},$$

we have that,

$$(Q^\star G_1^{-\nu_1 y^\star/\nu_2} G_1^{-\chi/\mu})^{\frac{\nu_2}{\nu_2 - y^\star - (\nu_1 y^\star + \chi\nu_2/\mu)(x^\star - x)}}$$

$$= (G_1^{y^\star\nu_1/\nu_2} G_1^{\frac{\nu_2+y^\star(\nu_1(x-x^\star)-1)}{\nu_2(\gamma+x^\star)}} G_1^{-\nu_1 y^\star/\nu_2} G_1^{\chi/\mu} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}} G_1^{-\chi/\mu})^{\frac{\nu_2}{\nu_2 - y^\star - (\nu_1 y^\star + \chi\nu_2/\mu)(x^\star - x)}}$$

$$= (G_1^{\frac{\nu_2+y^\star(\nu_1(x-x^\star)-1)}{\nu_2(\gamma+x^\star)}} G_1^{\frac{\chi(x-x^\star)}{\mu(\gamma+x^\star)}})^{\frac{\nu_2}{\nu_2 - y^\star - (\nu_1 y^\star + \chi\nu_2/\mu)(x^\star - x)}} = G_1^{1/(\gamma+x^\star)}.$$

Given this, $\mathcal{A}$ can break the q-sdh assumption as shown in [20] .

Therefore, in this case, assuming $\mathcal{A}'$ was successful, $\mathcal{A}$ succeeds with probability 1/2.

Therefore $\mathcal{A}$ solves the q-SDH problem with probability at least $\frac{\epsilon}{2n}$.

□

A.2 Soundness of Reputation

Assuming the CL signatures are existentially unforgeable under the chosen–message attack, the SPK is zero–knowledge and simulation sound extractable, and the random oracle model, our RS-GS construction satisfies soundness of reputation.

Proof. The CDL scheme makes use of Camenisch–Lysyanskaya (CL) signatures [39]. These are existentially unforgeable under the chosen–message attack [73] assuming the LRSW assumption [97], as detailed in Section 2.5.5.

We show that if an adversary $\mathcal{A}'$ exists, such that $\Pr[\mathbf{Exp}^{sound-rep}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}\ \mathsf{Aggr}) = 1] = \epsilon$, for some $\tau, \mathcal{R}, \hat{r}$, $\mathcal{U}, \mathsf{Aggr}$ with $|\mathcal{R}|$ polynomial in τ, $n = |\mathcal{U}|$ polynomial in τ, corrupting l users with the USK oracle, and ϵ is non–negligible in τ, then we can can build an adversary $\mathcal{A}$ that breaks existential unforgeability under the chosen–message attack for CL signatures. We describe $\mathcal{A}$ in Figure A.3. We then describe why the simulation given in Figure A.3 and the soundness of reputation experiment are indistinguishable to $\mathcal{A}'$ and how $\mathcal{A}$ works. The adversary $\mathcal{A}$ has input parameters $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2)$, the public key $(X = G_2^{\alpha}, Y = G_2^{\beta})$, and access to a CLSIGN oracle which takes input f and outputs $(a, a^{\beta}, a^{\alpha+f\alpha\beta})$, for $a \leftarrow\!\!\$\ \mathbb{G}_1$.

We first show that all inputs that $\mathcal{A}$ provides to $\mathcal{A}'$ are distributed identically to the soundness of reputation experiment.

Simulating inputs to $\mathcal{A}'$. (gpk, osk) are chosen in exactly the same way as in Setup. The POST and ALLREP oracles are the same as in the soundness of reputation experiment. Due to the distribution of outputs of the CLSIGN oracle, users' secret keys are of the form $(f, (A, A^{\beta}, A^{\alpha+f\alpha\beta}, A^{f\beta}))$ for randomly chosen $A \in \mathbb{G}_1$, therefore answers to USK queries are distributed identically to the soundness of reputation experiment.

If $b = 0$ or $uid \neq uid^*$ then the SENDFB oracle is the same as in the soundness of reputation experiment. We now show that outputs are correctly distributed if $b = 1$ and $uid = uid^*$. Let $A_2 = A_1^d$, because $A^* = A_1 A_2$, then

$$B^* = A_1^{\beta} A_2^{\beta} = A^{*\beta}, C^* = A_1^{\alpha} A_2^{\alpha} A_1^{\alpha\beta f} A_1^{\alpha\beta adf} = A^{*\alpha} (A^*)^{\alpha\beta(f+daf)/(1+d)},$$

$$D^* = (A_1^{\beta} A_1^{da\beta})^f = A^{*\beta(f+daf)/(1+d)} = B^{*(f+daf)/(1+d)}.$$

Therefore $((f + daf)/(1 + d), (A^*, B^*, C^*, D^*))$ is distributed identically to a user secret key, because f, d were chosen randomly and independently. Therefore, (A', B', C', D') are distributed identically to the SendFB oracle. Also $J = D^{*d''} = B^{*d''(f+daf)/(1+d)} = \mathcal{H}(I, r, t, \Omega)^{(f+daf)/(1+d)}$, which is distributed as in SendFB. The proof π can be simulated

$\mathsf{USK}(uid)$:

if $b = 1$ and $uid = uid^*$ $\mathcal{A}$ aborts
else $CU \leftarrow CU \cup \{uid\}$, **return** $\mathbf{usk}[uid]$

$\mathsf{POST}(I, uid, r, t, \omega)$:

return $\mathsf{PostItem}(gpk, I, (\mathbf{usk}_1[uid], \cdot), r, t, \omega)$

$\mathsf{SENDFB}(uid, fb, (I, r, t, \Omega))$:

if $b = 1$ and $uid = uid^*, Z \leftarrow \mathcal{H}(I, r, t, \Omega)$, for $((I, r, t, \Omega), Z, d'') \in HL, J \leftarrow D^{*d''}$
$d' \leftarrow_\$ \mathbb{Z}_p^*, A' \leftarrow A^{*d'}, B' \leftarrow B^{*d'}, C' \leftarrow C^{*d'}, D' \leftarrow D^{*d'}$
Simulate π with A', B', C', D', J **return** (A', B', C', D', J, π)
else return $\mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb)$

$\mathsf{ALLREP}(uid, t, r)$:

return $\mathsf{AllocateReputation}(gpk, (isk_1, \cdot), uid, (\mathbf{usk}_1[uid], \cdot), t, r, \mathcal{ID})$

$\mathcal{H}(\mathsf{in})$:

if $\exists(\mathsf{in}, \mathsf{out}, \cdot) \in HL$ **return** out
else $d'' \leftarrow_\$ \mathbb{Z}_p^*, HL \leftarrow HL \cup (\mathsf{in}, B^{*d''}, d'')$ **return** $B^{*d''}$

$\mathcal{A}^{\mathsf{CLSIGN}}(X, Y)$

Create empty lists $HL, CU, b \leftarrow_\$ \{0, 1\}, \mathcal{V} \leftarrow \mathcal{U}$
$gpk_2 \leftarrow (X, Y)$, finish computing $gpk, osk, isk_1, \mathbf{usk}_1, \mathbf{r}, \mathcal{ID}$ as in Setup
if $b = 1, uid^* \leftarrow_\$ \mathcal{U}, \mathcal{V} \leftarrow \mathcal{U} \backslash \{uid^*\}, f \leftarrow_\$ \mathbb{Z}_p^*, a \leftarrow_\$ \mathbb{Z}_p^*$ with $a \neq 1$
$(A_1, B_1, C_1) \leftarrow_\$ \mathsf{CLSIGN}(f), (A_2, B_2, C_2) \leftarrow_\$ \mathsf{CLSIGN}(af)$
$A^* \leftarrow A_1 A_2, B^* \leftarrow B_1 B_2, C^* \leftarrow C_1 C_2, D^* \leftarrow (B_1 B_2{}^a)^f$
$\forall uid \in \mathcal{V}$
$f_{uid} \leftarrow_\$ \mathbb{Z}_p^*, (A_{uid}, B_{uid}, C_{uid}) \leftarrow_\$ \mathsf{CLSIGN}(f_{uid}), D_{uid} \leftarrow B_{uid}^{f_{uid}}$
$\mathbf{usk}[uid] \leftarrow (\mathbf{usk}_1[uid], (f_{uid}, (A_{uid}, B_{uid}, C_{uid}, D_{uid})))$
$((I, r, t, \Omega), \{fb_j, \Phi_j\}_{j=1}^{l+1}) \leftarrow_\$ \mathcal{A}'^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP},\mathcal{H}}(gpk, osk)$
$j \leftarrow_\$ [l + 1]$, Extract f_j' for (fb_j, Φ_j)
Let $\Phi_j = (A', B', C', D', J, \pi)$
return (f_j', A', B', C')

Figure A.3: $\mathcal{A}$ which breaks existential unforgeability under the chosen–message attack for CL signatures, using $\mathcal{A}'$ which breaks soundness of reputation for our $\mathsf{RS\text{-}GS}$ construction

due to the zero–knowledge property of the SPK used. The hash oracle is distributed identically to the random oracle model, as d'' is chosen randomly for each query.

Reduction to existential unforgeability of CL signatures. We now show that the output of $\mathcal{A}$ is a valid forgery of a CL signature with non-negligible probability. For this to be the case, (f'_j, A', B', C') output by $\mathcal{A}$ should be a valid CL signature, and f'_j should not have been queried to the CLSIGN oracle. For all potential strategies a successful $\mathcal{A}'$ could take, we show that $\mathcal{A}$ is successful with non-negligible probability.

Assuming $\mathcal{A}'$ is successful, there is some $j^* \in [l+1]$ such that $f'_{j^*} \notin \{f_{uid} : uid \in \mathcal{C}\}$, where the f'_{j^*} was extracted from (fb_{j^*}, Φ_{j^*}). This is because all signatures output are valid and unlinkable. We assume $j^* = j$, which occurs with probability $1/(l+1)$.

If $\mathcal{A}'$ corrupts all users so that $l = n$, we assume $\mathcal{A}$ chose $b = 0$, which occurs with probability $1/2$, and so $\mathcal{A}$ does not abort. As $f'_{j^*} \notin \{f_{uid} : uid \in \mathcal{C}\}$, and $b = 0$, f'_{j^*} was not input to the CLSIGN oracle, and so $\mathcal{A}$ wins. Therefore $\mathcal{A}$ succeeds with probability $1/2(l+1)$.

If $\mathcal{A}'$ corrupts $l < n$ users, we assume $\mathcal{A}$ chose $b = 1$ and $uid^* \notin \mathcal{C}$, which occurs with probability $(n-l)/2n$. As uid^* is not queried to the USK oracle, $\mathcal{A}$ does not abort. For $\mathcal{A}$ to win, f_{j^*} must not have been input to the CLSIGN oracle.

If $f'_{j^*} = f_{uid}$ with $uid \in \mathcal{U} \backslash \mathcal{C}$, then we assume $uid = uid^*$, which occurs with probability $1/(n-l)$. As $a \neq 1$, $f'_{j^*} = \frac{f(1+da)}{1+d} \neq f$ or af, therefore f'_{j^*} was not input to CLSIGN. Therefore, $\mathcal{A}$ succeeds with probability $1/2n(l+1)$.

If $f'_{j^*} \notin \{f_{uid} : uid \in \mathcal{U}\}$, then the only case that f'_{j^*} was input to CLSIGN is if $f'_{j^*} = f$ or af. Let $f^* = \frac{f(1+da)}{1+d}$. Fixing f, a, then $g(\theta) = \frac{f(1+\theta a)}{1+\theta}$ is a bijective function, meaning that $f^* = \frac{f(1+da)}{1+d}$ is uniform and random and therefore independent of f, a. All inputs to $\mathcal{A}'$ were distributed independently of f, a, therefore $f'_{j^*} = f$ or af with probability $1/p$. Therefore, $\mathcal{A}$ succeeds with probability $\frac{(p-2)(n-l)}{2pn(l+1)}$.

Therefore, $\mathcal{A}$ succeeds with non-negligible probability.

□

A.3 Anonymity of Feedback

Assuming the DDH assumption in $\mathbb{G}_1$, the SPK is zero–knowledge and the random oracle model, our RS-GS construction satisfies anonymity of feedback.

Proof. We show that if an adversary $\mathcal{A}'$ exists, such that $\Pr[\mathbf{Exp}^{anon-fb-0}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1] - \Pr[\mathbf{Exp}^{anon-fb-1}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1] = \epsilon$, for some $\tau, \mathcal{R}, \hat{r}$, $\mathcal{U}, \mathsf{Aggr}$ with $|\mathcal{R}|$ and $n = |\mathcal{U}|$ polynomial in τ, q different items (I, r, t, Ω) are queried to the $\mathcal{H}$ and SENDFB oracles in the choose stage, and ϵ is non–negligible in τ, then we can can build an adversary $\mathcal{A}$ that breaks the DDH assumption. We describe $\mathcal{A}$ in Figure A.4. We then describe why the simulation given in Figure A.4 and the anonymity of feedback experiment are indistinguishable to $\mathcal{A}$, when a DDH tuple is input, and then how $\mathcal{A}'$ works.

Simulating the inputs to $\mathcal{A}'$. Assuming $\mathcal{A}$ is input a DDH tuple, inputs to $\mathcal{A}'$ are distributed identically to the anonymity of feedback experiment. If $\mathcal{A}$ does not abort, the USK, POST, and ALLREP oracles are exactly the same as in the anonymity of feedback experiment.

The SENDFB oracle is also the same as in the experiment, provided uid^* is not input. If uid^* is input, the output is distributed identically to the output of the oracle in the anonymity of feedback experiment. This is because letting $Q_2 = Q_1^{f_1}$, as $A' = Q_1^{d/\beta}$, then $B' = Q_1^d = A'^{\beta}, C' = A'^{\alpha} Q_2^{d\alpha} = A'^{\alpha} A'^{\beta\alpha f_1}, D' = Q_2^d = B'^{f_1}$.If $\mathcal{A}$ does not abort in SENDFB, then $j' \neq q^*$ or $b' = 1$, and so $\mathcal{H}(I, r, t, \Omega) = Q_1^{d'}$, and $J = Q_2^{d'} = \mathcal{H}(I, r, t, \Omega)^{f_1}$. Due to the zero–knowledge property of the SPK used, π can be simulated. Therefore the output of SENDFB is distributed identically to the anonymity of feedback experiment.

The hash oracle is distributed identically to the random oracle model, as d is chosen randomly. The (gpk, isk, osk) input in the choosing phase are chosen exactly as in Setup. The input in the guessing phase (A', B', C', D', J, π) is distributed identically to outputs of SendFB, because if $Q_4 = Q_3^{f_2}$ and $A' = Q_3^{d/\beta}$, $B' = Q_3^d = A'^{\beta}, C' = A'^{\alpha} Q_4^{d\alpha} = A'^{\alpha} A'^{\beta\alpha f_2}, D' = Q_4^d = B'^{f_2}$. If $\mathcal{A}$ does not abort, $\mathcal{H}(I^*, r^*, t^*, \Omega^*) = Q_3^{d'}$, and so $J = Q_4^{d'} = \mathcal{H}(I^*, r^*, t^*, \Omega^*)^{f_2}$. Again, due to the zero–knowledge property of the SPK used, π can be simulated. This signature is consistent with the SENDFB oracle, because $\mathcal{A}$ does not abort, so $uid_b = uid^*$, and (Q_1, Q_2, Q_3, Q_4) is a DDH tuple, therefore $f_1 = f_2$.

$\mathsf{USK}(uid)$:

if $uid = uid^*$, **if** $b = 0$ $\mathcal{A}$ **return** 1 **else** $\mathcal{A}$ **return** 0
else return $\mathbf{usk}[uid]$

$\mathsf{POST}(I, uid, r, t, \omega)$:

return $\mathsf{PostItem}(gpk, I, (\mathbf{usk}_1[uid], \cdot), r, t, \omega)$

$\mathsf{SENDFB}(uid, fb, (I, r, t, \Omega))$:

if $uid = uid^*$ **if** $(I, r, t, \Omega) = (I^*, r^*, t^*, \Omega^*)$ **if** $b = 0$ $\mathcal{A}$ **return** 1 **else** $\mathcal{A}$ **return** 0
$d \leftarrow\!\!\$\, \mathbb{Z}_p^*, A' \leftarrow Q_1^{d/\beta}, B' \leftarrow Q_1^d, C' \leftarrow A'^{\alpha} Q_2^{d\alpha}, D' \leftarrow Q_2^d, \quad S \leftarrow \mathcal{H}(I, r, t, \Omega)$
for $((I, r, t, \Omega), S, d', j') \in HL$ **if** $j' = q^*$ and $b' = 0$, **if** $b = 0$ $\mathcal{A}$ **return** 1 **else** $\mathcal{A}$ **return** 0
else $J \leftarrow Q_2^{d'}$, Simulate π with A', B', C', D', J
return (A', B', C', D', J, π)
else return $\mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb)$

$\mathsf{ALLREP}(uid, t, r)$:

return $\mathsf{AllocateReputation}(gpk, isk, uid, (\mathbf{usk}_1[uid], \cdot), t, r, \mathcal{ID})$

$\mathcal{H}(\mathsf{in})$:

if $\exists(\mathsf{in}, \mathsf{out}, \cdot, \cdot) \in HL$ **return** out
$j = j + 1$, **if** $j = q*$ and $b' = 0, d \leftarrow\!\!\$\, \mathbb{Z}_p^*, HL \leftarrow (\mathsf{in}, {Q_3}^d, d, j) \cup HL$, **return** ${Q_3}^d$
else $d \leftarrow\!\!\$\, \mathbb{Z}_p^*, HL \leftarrow (\mathsf{in}, {Q_1}^d, d, j) \cup HL$ **return** ${Q_1}^d$

$\mathcal{A}(Q_1, Q_2, Q_3, Q_4)$

1 : $b, b' \leftarrow\!\!\$\, \{0,1\}, uid^* \leftarrow\!\!\$\, \mathcal{U}, q^* \leftarrow\!\!\$\, [q], j \leftarrow 0,$ create empty list HL
2 : Compute $(gpk, isk, osk, \mathbf{usk})$ as in Setup, except $\mathbf{usk}_2[uid^*]$
3 : $(\mathsf{st}, uid_0, uid_1, fb, (I^*, r^*, t^*, \Omega^*)) \leftarrow\!\!\$\, \mathcal{A}'^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP},\mathcal{H}}(\mathsf{choose}, gpk, isk, osk)$
4 : **if** $uid^* \neq uid_b$, **return** 0
5 : **if** $b' = 0$, **if** $((I^*, r^*, t^*, \Omega^*), \cdot, \cdot, q^*) \notin HL$ **return** 0, **else** let $((I^*, r^*, t^*, \Omega^*), S, d', q^*) \in HL$
6 : **if** $b' = 1$, **if** $\exists((I^*, r^*, t^*, \Omega^*), \cdot, \cdot, \cdot) \in HL$ **return** 0
7 : $d' \leftarrow\!\!\$\, \mathbb{Z}_p^*, S \leftarrow Q_3^{d'}, j \leftarrow j + 1, q^* \leftarrow j, HL \leftarrow ((I^*, r^*, t^*, \Omega^*), S, d', q^*) \cup HL$
8 : $d \leftarrow\!\!\$\, \mathbb{Z}_p^*, A' \leftarrow Q_3^{d/\beta}, B' \leftarrow Q_3^d, C' \leftarrow A'^{\alpha} Q_4^{d\alpha}, D' \leftarrow Q_4^d$
9 : $J \leftarrow Q_4^{d'}$, Simulate π with A', B', C', D', J
10 : $b^* \leftarrow\!\!\$\, \mathcal{A}'^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP},\mathcal{H}}(\mathsf{guess}, \mathsf{st}, (A', B', C', D', J, \pi))$
11 : **if** uid_0 or uid_1 queried to the USK oracle $b^* \leftarrow 0$
12 : **if** $(uid_0, \cdot, (I, r, t, \Omega))$ or $(uid_1, \cdot, (I, r, t, \Omega))$ queried to the SENDFB oracle $b^* \leftarrow 0$
13 : **if** $b^* = b$ **return** 1, **else return** 0

Figure A.4: $\mathcal{A}$ which distinguishes between DDH tuples in $\mathbb{G}_1$, using $\mathcal{A}'$ which breaks anonymity of feedback for our RS-GS construction

Reduction to the DDH problem. If $\mathcal{A}$ was input a DDH tuple (Q_1, Q_2, Q_3, Q_4), then we assume uid^* was chosen so that $uid^* = uid_b$, which occurs with probability $1/n$, therefore $\mathcal{A}$ does not abort during line 4.

If $\mathcal{A}'$ outputs $(I^*, r^*, t^*, \Omega^*)$ in the choosing phase that they have queried to the hash oracle or the SENDFB oracle, we assume $b' = 0$ and that this was the q^*th such query, which occurs with probability $1/2q$. Then, provided $\mathcal{A}$ does not abort, all inputs to $\mathcal{A}'$ are distributed identically to the anonymity of feedback experiment.

The probability that $\mathcal{A}$ outputs 1 is then

$$1/2(\Pr[\mathbf{Exp}^{anon-fb-0}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 0] + \Pr[\mathbf{Exp}^{anon-fb-1}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1])$$

$$= 1/2(1 - \Pr[\mathbf{Exp}^{anon-fb-0}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1] + \Pr[\mathbf{Exp}^{anon-fb-1}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1])$$

$$= 1/2\epsilon + 1/2.$$

Therefore, $\mathcal{A}$ outputs 1 with probability $(\epsilon + 1)/4qn$.

If $\mathcal{A}'$ outputs $(I^*, r^*, t^*, \Omega^*)$ in the choosing phase that they have not queried to $\mathcal{H}$ or SENDFB, we assume $b' = 1$ is chosen which occurs with probability $1/2$, and so $\mathcal{A}$ never outputs early outputting 0. Then, by the same argument, $\mathcal{A}$ outputs 1 with probability $(\epsilon + 1)/4n$.

If (Q_1, Q_2, Q_3, Q_4) is not a DDH tuple, then $\mathcal{A}$ still aborts returning 0 in lines 4, 5, 6 with the same probability. $\mathcal{A}'$ is now given a signature in the guess stage that is independent of both f_{uid_0} and f_{uid_1}, therefore the probability $\mathcal{A}'$ guesses correctly is $1/2$. If $\mathcal{A}'$ queries USK with uid_0, uid_1 or SENDFB with uid_0, uid_1 and $(I^*, r^*, t^*, \Omega^*)$, then $\mathcal{A}$ outputs 1 with probability $1/2$. Therefore, if $\mathcal{A}'$ outputs $(I^*, r^*, t^*, \Omega^*)$ in the choosing phase that they have already queried to the hash oracle or the SENDFB oracle, then the probability $\mathcal{A}$ outputs 1 is $(1/2)(1/2qn) = 1/4qn$. If $\mathcal{A}'$ outputs $(I^*, r^*, t^*, \Omega^*)$ in the choosing phase that they have not queried to $\mathcal{H}$ or SENDFB, then the probability $\mathcal{A}$ outputs 1 is $(1/2)(1/2n) = 1/4n$.

Therefore $\mathcal{A}$ has at least a $\epsilon/4qn$ advantage in distinguishing between DDH tuples. □

$\mathsf{USK}(uid)$:

if $uid = uid^*$ $\mathcal{A}$ aborts
else return $\mathbf{usk}[uid]$

$\mathsf{POST}(I, uid, r, t, \omega)$:

return $\mathsf{PostItem}(gpk, I, (\mathbf{usk}_1[uid], \cdot), r, t, \omega)$

$\mathsf{SENDFB}(uid, fb, (I, r, t, \Omega))$:

if $uid = uid^*, d \leftarrow_\$ \mathbb{Z}_p^*, A' \leftarrow Q_1^{d/\beta}, B' \leftarrow Q_1^d, C' \leftarrow A'^{\alpha} Q_2^{d\alpha}, D' \leftarrow Q_2^d$
  $S \leftarrow \mathcal{H}(I, r, t, \Omega), ((I, r, t, \Omega), S, d') \in HL$
  $J \leftarrow Q_2^{d'}$, Simulate π with A', B', C', D', J
  return (A', B', C', D', J, π)
else return $\mathsf{SendFB}(gpk, \mathbf{usk}[uid], (I, r, t, \Omega), fb)$

$\mathsf{ALLREP}(uid, t, r)$:

return $\mathsf{AllocateReputation}(gpk, isk, uid, (\mathbf{usk}_1[uid], \cdot), t, r, \mathcal{ID})$

$\mathcal{H}(\mathsf{in})$:

if $\exists(\mathsf{in}, \mathsf{out}, \cdot) \in HL$ **return** out
else $d' \leftarrow_\$ \mathbb{Z}_p^*, HL \leftarrow HL \cup (\mathsf{in}, Q_1^{d'}, d')$ **return** $Q_1^{d'}$

$\mathcal{A}(Q_1, Q_2)$

Create empty list $HL, uid^* \leftarrow_\$ \mathcal{U}$
Compute $(gpk, isk, osk, \mathbf{usk})$ as in Setup, except $\mathbf{usk}_2[uid^*]$
$((I, r, t, \Omega), fb, \Phi) \leftarrow_\$ \mathcal{A}'^{\mathsf{USK},\mathsf{POST},\mathsf{SENDFB},\mathsf{ALLREP},\mathcal{H}}(gpk, isk, osk)$
Extract f^* from Φ **return** f^*

Figure A.5: $\mathcal{A}$ which breaks the DL problem in $\mathbb{G}_1$, using $\mathcal{A}'$ which breaks non–frameability for our RS-GS construction

A.4 Non–frameability

Assuming the DL assumption in $\mathbb{G}_1$, the SPK is zero–knowledge and simulation sound extractable and the random oracle model, our RS-GS construction satisfies non–frameability.

Proof. We show that if an adversary $\mathcal{A}'$ exists, such that $\Pr[\mathbf{Exp}^{non-frame}_{\mathcal{A}',\mathsf{RS\text{-}GS}}(\tau, \mathcal{R}, \hat{r}, \mathcal{U}, \mathsf{Aggr}) = 1] = \epsilon$, for some $\tau, \mathcal{R}, \hat{r}$, $\mathcal{U}, \mathsf{Aggr}$ with $|\mathcal{R}|$ polynomial in τ, $n = |\mathcal{U}|$ polynomial in τ, and ϵ is non–negligible in τ, then we can can build an adversary $\mathcal{A}$ that breaks the DL problem. We describe $\mathcal{A}$ in Figure A.5. We then describe why the simulation given in Figure A.5 and the non–frameability experiment are indistinguishable to $\mathcal{A}$, and how $\mathcal{A}'$ works.

We first show that all inputs $\mathcal{A}$ provides to $\mathcal{A}'$ are distributed identically to the non–frameability experiment.

Simulating inputs to $\mathcal{A}'$. (gpk, isk, osk) are chosen in exactly the same way as in Setup. Provided $\mathcal{A}$ does not abort, the USK, POST and ALLREP oracles are the same as in the non–frameability experiment. If $uid \neq uid^*$ is input, the SENDFB oracle is identical to the non–frameability experiment. If $uid = uid^*$ is input, letting $Q_2 = Q_1^f$, and as $A' = Q_1^{d/\beta}$, then $B' = Q_1^d = A'^{\beta}, C' = A'^{\alpha} Q_2^{d\alpha} = A'^{\alpha} A'^{\beta\alpha f}, D' = Q_2^d = B'^f$. Because $\mathcal{H}(I, r, t, \Omega) = Q_1^{d'}$, $J = Q_2^{d'} = \mathcal{H}(I, r, t, \Omega)^f$. π can be simulated due to the zero–knowledge property of the SPK used. Therefore the output of SENDFB is distributed identically to the anonymity of feedback experiment. The hash oracle is distributed identically to the random oracle model, because d' is chosen randomly.

Reduction to the DL problem. Assuming $\mathcal{A}'$ is successful, there is some $uid \notin \mathcal{C}$ such that $(I, r, t, \Omega), fb', \Phi')$ was output by the SENDFB oracle under input uid, $((I, r, t, \Omega), fb$, $\Phi)$ was not output by SENDFB under input uid, and $\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb, \Phi, fb'$, $\Phi') = 1$. The signature Φ output by $\mathcal{A}'$ was not output by SENDFB with input user $uid' \neq uid$, because otherwise $\mathsf{LinkFB}(gpk, (I, r, t, \Omega), fb, \Phi, fb', \Phi') \neq 1$. Therefore, it is possible to extract f^*. We assume $uid = uid^*$, which occurs with probability $1/n$. Then $\mathcal{A}$ will not abort. Again, let $Q_2 = Q_1^f$, and also let $((I, r, t, \Omega), S, \cdot) \in HL$, and $\Phi = (A^*, B^*, C^*, D^*, J^*, \pi^*)$, $\Phi' = (A', B', C', D', J, \pi)$. As the signatures are linked, $J = S^f = S^{f^*} = J^*$, therefore $f = f^* \mod p$. This means $\mathcal{A}$ successfully finds the discrete logarithm with probability ϵ/n. □